Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Физико-технологический институт

И. Г. НЕУДАЧИН

ТАБЛИЦЫ DELPHI ДЛЯ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Учебно-методическое пособие

Министерство образования и науки Российской Федерации
Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

И. Г. Неудачин

ТАБЛИЦЫ DELPHI ДЛЯ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Учебно-методическое пособие

Рекомендовано методическим советом УрФУ
для студентов, обучающихся по направлениям подготовки
«Информатика и вычислительная техника»,
«Ядерные физика и технологии»

Екатеринбург
Издательство Уральского университета
2016

УДК 004.65(075.8)
ББК 32.972.34я73
Н57

Рецензенты:
кафедра прикладной информатики УралГАХА (завкафедрой канд. техн. наук, доц. Г. Б. Захарова);
замдиректора по производству ООО «Наудок» канд. физ.-мат. наук, доц. О. А. Евсегнеев

Научный редактор — канд. физ.-мат. наук, проф. В. И. Рогович

На обложке использовано изображение с сайта https://www.contactcenterworld.com/imagestemp/592015175448_shutterstock_157231454.jpg

Неудачин, И. Г.

Н57 Таблицы Delphi для управления базами данных : учеб.-метод. пособие / И. Г. Неудачин. — Екатеринбург : Изд-во Урал. ун-та, 2016. — 96 с.

ISBN 978-5-7996-1790-5

В учебно-методическом пособии предлагается руководство по визуальному программированию систем управления базами данных на языке Object Pascal в среде Delphi. Пособие содержит необходимые сведения по обработке таблиц баз данных и файлов при помощи визуальных компонентов. Описаны структуры графических интерфейсов, их настройка и технология кодирования обработки событий. Предлагаются задания для самостоятельной работы.

Методическое пособие предназначено для студентов и преподавателей, использующих базы данных на персональных ЭВМ.

Библиогр.: 5 назв. Табл. 2. Рис. 37. Прил. 3.

УДК 004.65(075.8)
ББК 32.972.34я73

ISBN 978-5-7996-1790-5

Глава 1.
Архитектура баз данных в Delphi

1.1. Принципы строения баз данных

Реляционная база данных (БД) — это набор таблиц. Таблица является набором характеристик некоторого множества объектов. Столбцы таблицы соответствуют характеристикам объектов, полям. Строки, таким образом, содержат записи, соответствующие отдельным объектам. В программном проекте таблица представлена визуальным компонентом Table.

Каждое поле обладает именем (идентификатором) и типом хранящихся в нем данных. Имя поля нужно для ссылок на него.

Непротиворечивость информации в таблице обеспечивается введением ключевых полей. Ключевое поле обеспечивает своим значением уникальность каждой записи. Ключевым может быть одно или несколько полей. Например, ключевым может быть поле порядкового номера записи.

При работе с таблицей программа имеет дело с одной текущей записью. На нее указывает курсор, положение которого может меняться для обработки другой записи. Записи в таблице физически могут располагаться произвольно (в порядке их ввода). Но когда данные предъявляются пользователю, они должны быть упорядочены. Для упорядочивания данных по полям используется понятие индекса. Он показывает последовательность просмотра таблицы. Это посредник между пользователем и таблицей.

Курсор-указатель скользит по индексу. Индекс указывает (рис. 1.1) на запись таблицы. Для пользователей таблица выглядит упорядоченной.

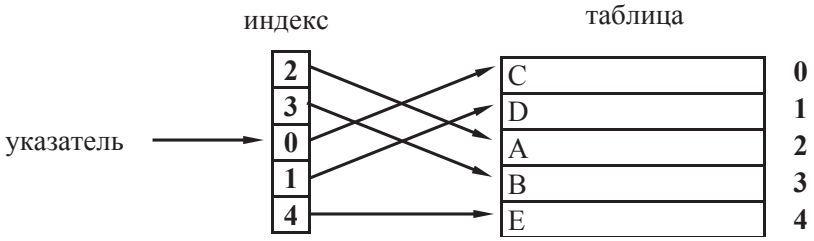


Рис. 1.1. Индексирование поля таблицы

Пользователь может сменить индекс и последовательность просмотра изменится. Сама таблица не перестраивается физически. Меняется только индекс, т. е. последовательность ссылок на записи.

Индексы могут быть первичными (по ключевому полю) и вторичными (по другому полю). Вторичные индексы создаются по другим полям таблицы при ее создании или позднее, при реструктуризации. Если индекс состоит из нескольких полей, то он называется сложным. В этом случае происходит упорядочивание базы данных по первому полю, затем внутри групп записей с одинаковым первым полем — по второму полю и т. д.

База данных обычно содержит не одну, а несколько таблиц, т. к. их совокупность дает больше информации. Создают базы данных и обслуживают информационные запросы к ним системы управления базами данных (СУБД): Paradox, dBase, MsAccess, FoxPro, Oracle, InterBase, Sybase и др.

Разные СУБД по-разному организуют и хранят БД. Например, Paradox и dBase используют для хранения каждой таблицы отдельный файл. Здесь БД — это каталог с файлами таблиц. В MsAccess и InterBase несколько таблиц хранятся в одном файле. В этом случае БД — это имя файла с путем доступа к нему. Системы типа клиент/сервер (Sybase, MsSQL) хранят все данные на отдельном компьютере — сервере. Они общаются с клиентом на специальном языке SQL.

Свойства БД разнообразны: типы СУБД, каталоги, файлы, серверы. Здесь сложно учитывать изменения, т. к. программу надо переделывать. Проблему решают псевдонимы БД. Псевдоним (alias) содержит всю информацию для доступа к БД. Эта информация сообщается разработчиком только один раз при создании псевдонима.

1.2. Типы баз данных

Задачи разного масштаба целесообразно решать, используя соответствующие модели БД. Процесс определения подходящей модели БД для конкретного проекта приложения называется масштабированием. Есть четыре модели БД:

- автономные (локальные);
- файл-серверные;
- клиент/сервер;
- многоярусные (распределенные).

Изучим приемы работы, нужные для любых из перечисленных моделей.

Отметим, что работа с данными в Delphi происходит через Borland Database Engine (BDE). Это процессор БД фирмы Borland. Соответствующая программа должна быть поставлена на ЭВМ пользователя во всех моделях, кроме многоярусных моделей.

1.3. Основные понятия баз данных

Таблица — набор строк (записей) в БД.

Строка — поле (атрибут), которое содержится в каждой строке таблицы.

Borland Database Engine (BDE) — набор модулей DLL и других файлов. С их помощью Delphi и другие продукты Borland обращаются к БД.

Драйвер — вспомогательная программа для общения с БД данного типа.

Драйвер BDE — модуль или набор модулей DLL для взаимодействия с конкретной СУБД.

Псевдоним BDE. Программы Delphi обращаются к драйверам BDE через псевдонимы (alias) BDE. Alias обозначает набор параметров для подключения BDE к БД. Создают псевдоним отдельным приложением BDE Administrator, Delphi Database Explorer или Database. Драйвер нужен для СУБД, alias — для обращения к конкретной БД этой СУБД.

Драйвер SQL Links обеспечивает доступ BDE к БД из СУБД архитектуры клиент/сервер.

Драйверы ODBC — доступ на основе спецификации Open Database Connectivity.

1.4. Доступ к данным

Компоненты доступа к данным часто относятся к невидимым и расположены на странице Data Access палитры визуальных компонентов среды визуального программирования Delphi: TBatabase, TTable, TDataSource.

TdataSet — класс Delphi для доступа к таблицам и табличным запросам. Компоненты TTable, TQuery, TSortedProc являются потомками TDataSet и относятся к его семейству (рис. 1.2).

TTable — компонент доступа к таблице базы данных. Свойство TableName содержит имя файла реальной таблицы с псевдонимом (путь к файлу) DatabaseMame.

TQuery создает, выполняет и обрабатывает запросы SQL.

TDataSource связывает компоненты семейства TDataSet и элементы управления для работы с базами данных.

Элементы управления данными. Визуальные компоненты страницы DataControls палитры компонентов. К ним относятся специализированные стандартные элементы, умеющие работать с базами данных: TDBEdit, TDBMemo, TDBImage, TDBGrid, TDBNavigator.

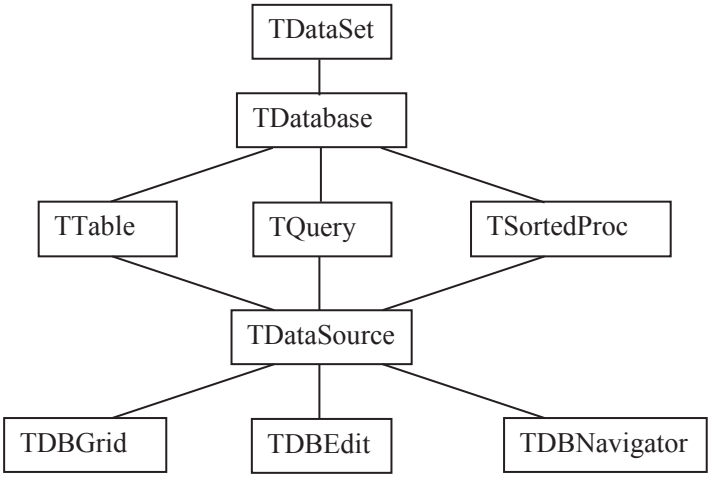


Рис. 1.2. Классы доступа к базам данных

TField — класс Delphi доступа к полям таблицы базы данных. Редактор полей таблицы создает порожденные классы: TStringField, TIntegerField.

Компонент TDatabase не используется для доступа к базе данных, но имеет ряд дополнительных свойств и элементов управления, удобных для разработки приложений.

1.5. Связь с базой данных в Delphi

Приложение Delphi обращается к БД через BDE (рис. 1.3) и сообщает псевдоним и таблицу. BDE по псевдониму находит драйвер соответствующей СУБД, обрабатывает запрос пользователя и возвращает в приложение результаты обработки. BDE обеспечивает доступ с СУБД: Paradox, dBase, MsAccess, FoxPro.

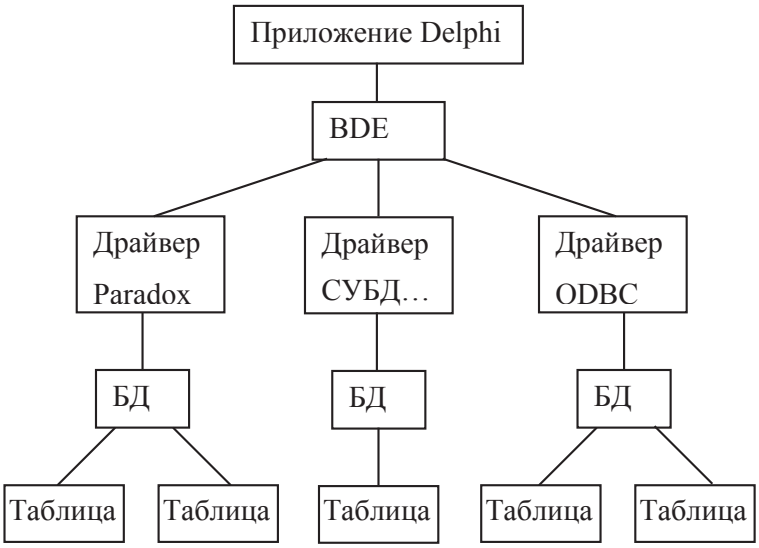


Рис. 1.3. Схема связи приложения с БД

Если в BDE нет драйвера нужной СУБД, то используется драйвер ODBC. Microsoft создала драйвер ODBC практически к любым СУБД. Если собственный драйвер соответствующей СУБД есть, то BDE связывается через него с БД и с нужной таблицей в ней, обрабатывает запрос и возвращает в приложение результаты обработки.

BDE поддерживает стандартный язык запросов SQL. Он позволяет обмениваться данными с SQL-серверами (Sybase, MsSQL, Oracle, InterBase). SQL используется при работе на платформе клиент/сервер.

1.6. Класс TTable (таблица)

Приведем основные факты, которые необходимо знать, прежде чем начать разрабатывать программы, работающие с базами данных.

1.6.1. Основные компоненты

Изучив этот раздел, нужно понять большинство механизмов доступа к данным, которые есть в Delphi. Более подробно здесь рассказывается о классах TTable и TDataSource.

Имеются несколько основных компонентов, которые вы будете использовать постоянно для доступа к БД. Эти объекты могут быть разделены на три группы в зависимости от интерфейса связи с БД:

- невизуальные: TTable, TQuery, TDataSet, Tfield;
- визуальные: TDBGrid, TDBEdit;
- связующие: TdataSource.

Первая группа включает невизуальные классы, которые используются для управления таблицами и запросами. Эта группа сосредотачивается вокруг компонент типа TTable, TQuery, TDataSet и TField. В Палитре Компонентов эти объекты расположены на странице Data Access.

Вторая важная группа классов — визуальные, которые показывают данные пользователю и позволяют ему просматривать и модифицировать их. Эта группа классов включает компоненты типа TDBGrid, TDBEdit, TDBImage и TDBComboBox. В Палитре Компонентов эти объекты расположены на странице Data Controls.

Имеется и третий тип, который используется для того, чтобы связать предыдущие два типа объектов. К третьему типу относится только невизуальный компонент TDataSource.

1.6.2. Класс TDataSet

TDataSet класс — один из наиболее важных объектов БД. Для того чтобы начать работать с ним, нужно взглянуть на следующую иерархию.

TDataSet содержит абстрактные методы там, где должно быть непосредственное управление данными. TDBDataSet знает, как обращаться с паролями и что нужно сделать, чтобы присоединить вас к определен-

ной таблице. TTable знает (т. е. уже все абстрактные методы переписаны), как обращаться с таблицей, ее индексами и т. д.

```
TDataSet
        |
    TDBDataSet
        |
        |-- TTable
        |-- TQuery
        |-- TStoredProc
```

Как вы увидите далее, TQuery имеет определенные методы для обработки SQL запросов.

TDataSet — инструмент, который вы будете использовать для того, чтобы открыть таблицу и перемещаться по ней. Конечно, вы никогда не будете непосредственно создавать объект типа TDataSet. Вместо этого вы будете использовать TTable, TQuery или других потомков TDataSet (например, TQBE). Полное понимание работы системы и точное значение TDataSet будут становиться все более ясными по мере прочтения этого раздела.

На наиболее фундаментальном уровне Dataset — это просто набор записей с указателями начала BOF и конца EOF набора. Любой Dataset состоит из ряда записей (каждая запись содержит N полей) и есть указатель на текущую запись.

В большинстве случаев dataset будет иметь прямое, один к одному, соответствие с физической таблицей, которая существует на диске. Однако в других случаях вы можете исполнять запрос или другое действие, возвращающее dataset, который содержит либо любое подмножество записей одной таблицы, либо объединение (join) между несколькими таблицами. В тексте будут иногда использоваться термины DataSet и TTable как синонимы.

Обычно в программе используются объекты типа TTable или TQuery, поэтому в следующих нескольких главах будет предполагаться существование объекта типа Ttable, называемого Table1.

Итак, самое время начать исследование TDataSet. Как только вы познакомитесь с его возможностями, начнете понимать, какие методы использует Delphi для доступа к данным, хранящимся на диске в виде БД. Ключевой момент здесь — не забывать, что почти всякий раз, когда программист на Delphi открывает таблицу, он будет использовать TTable или TQuery, которые являются просто некоторой надстройкой над TDataSet.

1.6.3. Открытие и закрытие DataSet

Если вы используете TTable для доступа к таблице, то при открытии данной таблицы заполняются некоторые свойства TTable (количество записей RecordCount, описание структуры таблицы и т. д.).

Прежде всего во время дизайна необходимо поместить на форму объект TTable и указать, с какой таблицей БД хотите работать. Для этого нужно заполнить в Инспекторе Объектов свойства DatabaseName и TableName. В DatabaseName можно указать директорию, в которой лежат таблицы БД формата dBase или Paradox (например, C:\DELPHI\DEMOS\DATA), либо выбрать из списка псевдоним базы данных (например, DBDEMOS). Псевдоним базы данных (Alias) определяется в утилите Database Engine Configuration. Теперь если свойство Active установить в True, то при запуске приложения таблица будет открываться автоматически.

Есть два различных способа открыть таблицу во время выполнения программы. Можно написать следующую строку кода:

```
Table1.Open;
```

Или можете установить свойство Active равным True:

```
Table1.Active := True;
```

Нет никакого различия между результатом, производимым этими двумя операциями. Метод Open, однако, сам заканчивается установкой свойства Active в True, так что, может быть, даже чуть более эффективно использовать свойство Active напрямую.

Так же, как имеются два способа открыть таблицу, так есть два способа закрыть ее. Самый простой способ — просто вызывать Close:

```
Table1.Close;
```

Или, если вы желаете, можно написать:

```
Table1.Active := False;
```

Еще раз повторим, что нет никакой существенной разницы между двумя этими способами. Нужно только помнить, что Open и Close — это методы (процедуры), а Active — свойство.

1.6.4. Свойства компонента TTable

Иерархия наследования объекта: Tobject, Tpersistent, Tcomponent, TdataSet, TBDEDataSet, TDBDataSet.

TTable — компонент набора данных Dataset, который формирует таблицу базы данных. Компонент находится в палитре компонентов (ПК) на странице BDE и описан в модуле dbtables.

TTable используется для того, чтобы обратиться к данным в отдельной таблице базы данных, используя Borland Database Engine — BDE (процессор баз данных фирмы BORLAND). TTable обеспечивает прямой доступ к каждой записи и полю в основной таблице базы данных из СУБД (Paradox, dBASE, Access, FoxPro, ODBC-совместимой базы данных) или базы данных SQL на удаленном сервере (типа InterBase, Oracle, Sybase, MS-SQL Server, Informix или DB2).

TDataSet.Name

Обозначает имя набора данных, на которое ссылаются другие компоненты.

```
property Name: TComponentName;
```

Описание

Используйте имя Name, для того чтобы изменить имя набора данных и отразить его цель в текущем приложении. По умолчанию IDE назначает последовательные имена на основе типа компонента вида 'Table1', 'Table2' и т. д.

Если имя набора данных изменено во время проектирования, то любой из компонентов поля, использующий имя набора данных как префикс к имени поля, также модифицируется.

TDataSet.Active

Определяет, открыт или закрыт набор данных.

```
property Active: Boolean;
```

Описание

Используйте Active, для того чтобы определить или подключить набор данных к данным в базе данных. Когда Active равно False, набор данных закрыт; набор данных не может читать данные из БД или писать данные в нее. Когда Active равно True, данные могут читаться из и записываться в БД.

Установка в Active значения True:

- вызывает драйвер события BeforeOpen, если он определен для набора данных;

- устанавливает состояние набора данных dsBrowse;
- открывает курсор в наборе данных;
- вызывает драйвера события After Open, если он определен для набора данных.

Если происходит ошибка при открытии набора данных, устанавливается состояние набора данных dsInactive, и курсор закрывается.

Приложение должно установить в Active значение False перед изменением других свойств, которые действуют на состояние базы данных, или средств управления, которые отображают данные в приложении.

Обратите внимание: вызов метода Open устанавливает в Active значение True; вызов метода Close устанавливает в Active значение False.

DBDataSet.DatabaseName

Определяет alias базы данных, связанной с этим набором данных.

```
property DatabaseName: String;
```

Описание

Используйте DatabaseName, чтобы определить alias базы данных или путь, связанный с этим компонентом набора данных. DatabaseName должно соответствовать имени компонента базы данных, используемого в программе.

Обратите внимание: попытка установить DatabaseName, когда база данных, связанная с этим компонентом, открыта, вызывает исключительную ситуацию.

Совет: во время проектирования щелчок по списку значений справа от DatabaseName выведет возможные для выбора существующие базы данных, для того чтобы установить DatabaseName.

TTable.TableName

Указывает имя таблицы базы данных, которую этот компонент представляет.

```
property TableName: TFileName;
```

Описание

Используйте TableName, для того чтобы определить имя таблицы базы данных, которую подключает этот компонент. Для того чтобы установить в TableName правильное значение, свойство DatabaseName должно быть уже установлено. Если DatabaseName установлено во время проектирования, то выберите имя таблицы TableName из раскрывающегося списка в инспекторе объектов.

Обратите внимание: для того чтобы определять TableName, свойство Active должно быть равно False.

TTable.TableType

Указывает структуру таблицы базы данных для таблицы, которую этот компонент представляет.

```
type TTableType = (ttDefault, ttParadox, ttDBase,
ttASCII, ttFoxPro);
property TableType: TTableType;
```

Описание

Используйте TableType, для того чтобы определить структуру таблицы базы данных для dBASE, Paradox, FoxPro или таблицы ASCII. TableType не обращается к таблицам на удаленных SQL серверах. В TableType может быть задано любое из следующих значений:

- ttDefault (значение по умолчанию) определяет, что напечатается основанный на расширении файл для таблицы;
- ttParadox — таблица Paradox;
- ttDBase — таблица dBASE;
- ttFoxPro — таблица FoxPro;
- ttASCII — таблица — текстовый файл с разграниченными запятой строками для каждого поля, заключенными в апострофы.

Если в TableType установлено значение ttDefault, то расширение имени файла таблицы определяет тип таблицы:

- DB или none — таблица Paradox;
- DBF — таблица dBASE;
- TXT — таблица ASCII.

Если TableType = ttParadox, ttDBase, ttFoxPro или ttASCII, то тип таблицы не будет зависеть от расширения имени файла таблицы.

Глава 2.
Быстрая разработка приложений БД

Воспользуемся в простых приложениях готовыми примерами баз данных Delphi из каталога примеров C:\Program Files\Common Files\Borland Shared\Data\. Таблицу базы данных можно создать специальным инструментом Delphi — БД рабочего стола Database Desktop (DbD). Доступ к DbD возможен через системное меню Все программы|Borland Delphi 7 | Database Desktop или из главного меню среды Delphi Tools| Database Desktop.

2.1. Использование Database Desktop

Задание 1. Используйте Database Desktop (DbD) для создания структуры новой таблицы.

1. Войдите через главное меню Delphi в DbD:

Tools| Database Desktop.

Откроется окно, содержащее меню работы с базами данных:

File Edit Tools Window Help.

2. Опишем структуру (спецификацию) таблицы базы данных:

File| New| Table.

Появится диалоговое окно запроса типа создаваемой таблицы. Выберем тот тип, что проще — Paradox 7. Нажмите кнопку (кн.) OK.

3. Появится другое окно диалога (Field roster) для задания свойств полей и таблицы. Задайте эти свойства.

	Field Name	Type	Size	Key
1	Number	+		*
2	Name	A	20	
3	Year	S		
4	Department	A	10	

Список типов полей для выбора появится, если в колонке Type нажать правую кнопку мыши или клавишу «пробел». Типы полей в колонке Type задаются подчеркнутым символом в названии типа: Alpha (текстовый), Number (действительный), $ Money, Short (-32767..32767), Long Integer, Date, Time, Memo, Graphic, Logical, Autoincrement{+} (счетчик).

Ширина поля Size указывается в символах для строк, если это требуется. Рекомендуется, чтобы в таблице одно поле было ключевым (Key), т.е. уникальным для каждой записи. Здесь ключевым является поле Number.

4. Сохраните информацию.

Нажмите кн. Save As...| Появится окно диалога «Save Table as». Здесь в окне ввода «Drive (or alias)» введите имя псевдонима DBDEMOS| В окне ввода «New file name» укажите имя файла таблицы studx.db| кн. OK.

Задание 2. Заполните таблицу данными в Database Desktop (DbD).

1. В окне DbD откройте созданную таблицу для заполнения данными:

File| Open| Table...

Откроется диалоговое окно выбора таблицы.

В строке Alias: DBDEMOS |▼ выберите из списка псевдонимов DBDEMOS.

Откроется папка DATA. Найдите в папке свою таблицу studx.db и щелкните по ней, чтобы указать свой выбор. Нажмите кн. «Открыть».

2. Появится пустая таблица в своем окне и панель инструментов. Расширится меню DbD: File Edit View Table Record Tools Window Help. Выберите в меню Table|Edit Data.

Наполните таблицу данными.

studx	Number	Name	Year	Department
	1	Иванов И.И.	1996	ФТ
	2	Петров П.П.	1997	МТ
	3	Сидоров С.С.	1998	МФ

Учтите, что последняя запись сохранится только после перехода на следующую строку таблицы. Возможно, понадобится настройка на кириллицу.

Edit| Preferences... В окне диалога выберите страницу General. В рамке Default system font нажмите кн. Change... | выберите шрифт Font:

Courier New Cyr| OK. Перезапустите DBD: File|Close и File|Exit, затем Tools|Database Desktop и т. д.

3. Закончите работу в DBD: File| Close и File|Exit.

Задание 3. Заполните таблицу данными из Delphi.

Построим приложение с доступом к таблице.

1. Войдите в Delphi и создайте новый проект приложения:

File| New| Application.

2. Сохраните файлы проекта приложения под новыми именами в своей папке: File|Save Project As... В диалоговом окне выберите свою папку и разместите в ней файлы, назвав их, например, Unstudx (модуль) и Prjstudx (файл проекта).

3. В Палитре Компонентов (ПК) выберите страницу BDE (Доступа к данным), чтобы воспользоваться компонентами этой страницы палитры. Вы обнаружите компонент Table (Таблица) в левой части страницы. Когда вы найдете компонент Table, нажмите его один раз, чтобы выбрать, затем нажмите мышью на форме, чтобы поместить (опустить) компонент на форму.

Когда вы поместите таблицу на форму, Delphi назовет объект Table1 по умолчанию.

4. В Инспекторе Объектов (ИО) задайте реквизиту DatabaseName объекта Table1 значение DBDEMOS. DBDEMOS — псевдоним базы данных, которая содержит таблицу, которую вы собираетесь использовать. Вы можете выбрать DBDEMOS из раскрывающегося списка реквизита DatabaseName. Установка этого значения реквизита обеспечивает доступ к таблице базы данных.

Выберите свойство TableName = studx.db из предложенного списка.

Установите свойство Active = true. Это откроет таблицу при пуске программы. Задайте свойство Name = studx.

Теперь у программы есть вся информация для подключения к таблице, но нет способов манипуляций данными таблицы.

Задание 4. Установите компонент связи для будущего управления таблицей данных.

1. В ПК со страницы Data Access поместите компонент DataSource на форму.

2. В ИО установите свойства объекта DataSource:

DataSet = studx;

Name = dsStud.

Задание 5. Задайте управление для просмотра и модификации данных в таблице.

1. В ПК со страницы Data Controls поместим на форму компонент DBGrid (сетка). Это первый видимый при исполнении программы объект. Сделайте его больше, для того чтобы увидеть поля данных.

2. В ИО установите для объекта DBGrid свойство DataSource = dsStud. Это свяжет сетку с источником данных.

3. На сетке автоматически появились имена полей и колонки с информацией, если она была занесена в таблицу.

4. File| Save All.

5. Run| Run. Приложение запущено. Смотрите, редактируйте, дополняйте таблицу данными. Завершите программу кнопкой ☒.

Задание 6. Добавьте еще один орган управления DBEdit. Он отображает одно поле текущей записи таблицы.

1. В ПК со страницы Data Controls поместите на форму компонент DBEdit.

2. В ИО установите свойства DBEdit: DataSource = dsStud; DataField = Name. Окошко редактирования DBEdit будет показывать поле с фамилией из вашей таблицы.

3. File| Save All.

4. Run| Run. Приложение запущено. Перемещение по записям происходит с помощью полосы прокрутки. Завершите программу кнопкой ☒. В больших таблицах удобнее использовать навигатор.

Задание 7. Добавьте в форму проекта приложения навигатор DBNavigator.

1. В ПК со страницы Data Controls поместим на форму компонент DBNavigator.

2. В ИО установите свойства DBNavigator: DataSource = dsStud; ShowHint = True. Последнее обеспечивает появление подсказки, когда курсор мыши будет задерживаться на кнопке панели управления. Видимость кнопок определяет свойство навигатора VisibleButtons.

4. File| Save All.

5. Run| Run. Просмотрите данные, используя навигатор.

2.2. Перетаскивание полей из редактора полей таблицы на форму

1. Создайте новый проект File|New Application.

2. Сохраните проект в новой отдельной папке File|Save Project As...

3. В палитре компонентов, со страницы BDE поместите Table на форму.

4. Выберите Table на форме. В Инспекторе Объектов (ИО) задайте свойства компонента

DatabaseName = DBDEMOS из списка

TableName = BIOLIFE.DB из списка

5. Два щелчка левой кнопкой мыши (л. к. м.) на компоненте Table формы откроют окно Редактора Полей Таблицы (РПТ). Один щелчок правой кнопки мыши (п. к. м.) на РПТ откроет локальное контекстное меню. Выберите в меню команду Add fields...

Откроется диалоговое окно. Выберите выделением все поля базы данных и нажмите кнопку OK. Все поля БД окажутся в РПТ.

6. Выбирайте по очереди поля в РПТ и в ИО задавайте их свойство

Display Label = название поля по-русски.

7. Вновь выберите все поля в РПТ, захватите их л. к.м. и перетащите их на форму. Компоненты автоматически подбираются в соответствии с типом поля.

8. Выберите Table на форме. В Инспекторе Объектов (ИО) задайте свойство компонента, для того чтобы открыть БД.

Active = True

Продумайте дизайн и разместите все элементы базы данных на форме в соответствии со своим замыслом.

9. В ПК со страницы Data Controls поместите DBNavigator на форму.

10. Выберите DBNavigator на форме. В ИО задайте свойство компонента

DataSource = DataSource1

11. Все сохраните File|Save All.

12. Компилируйте и выполните приложение Run|Run.

13. Тестирование и отладка.

14. Вставьте в базу данных дополнительное описание какой-либо рыбы.

2.3. Пример перетаскивания полей из редактора на форму

Разместите на форме компоненты Table и DBNavigator в соответствии с рис. 2.1. Свяжите Table с БД. Задайте свойства полей таблицы из следующего списка свойств. Перетащите поля из редактора на форму.

2.3.1. Список свойств полей таблицы

```
object Table1Number: TAutoIncField
  DisplayLabel = '№'
  FieldName = 'Number'
end
object Table1Name: TStringField
  DisplayLabel = 'Фамилия'
  FieldName = 'Name'
end
object Table1Year: TSmallintField
  DisplayLabel = 'Год окончания'
  FieldName = 'Year'
end
object Table1Department: TStringField
  DisplayLabel = 'Факультет'
  FieldName = 'Department'
  Size = 10
end
object Table1Degree: TBooleanField
  DisplayLabel = 'Звание'
  FieldName = 'Degree'
end
object Table1Photo: TGraphicField
  DisplayLabel = 'Фото'
  FieldName = 'Photo'
  BlobType = ftGraphic
  Size = 240
end
```

2.3.2. Форма и редактор полей

Пример размещения полей на форме (рис. 2.1).

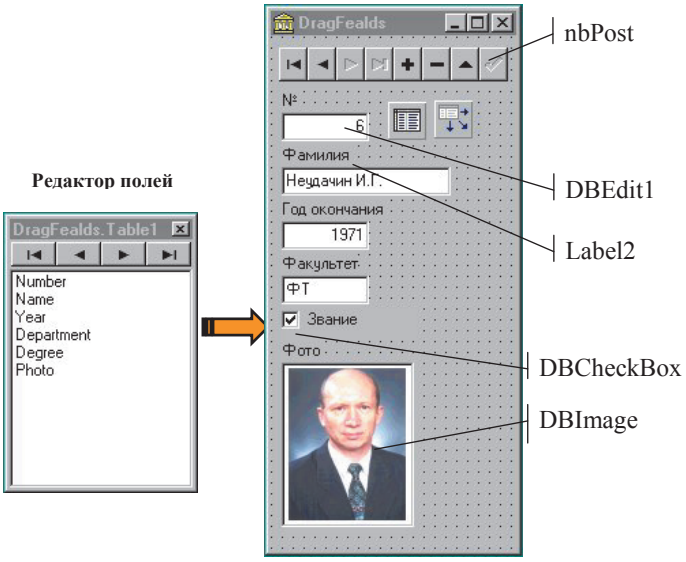


Рис. 2.1. Форма и окно редактора полей

2.4. Мастер баз данных

Мастер баз данных позволяет быстро создавать простые приложения. Он вызывается из меню среды Delphi.

1. File|New Application.
2. Database|Form Wizard... ->.

Откроется диалоговое окно.

Тип формы: Create a simple form| Create a form using TTable objects.

3. Нажмите кн. Next >.

Откроется диалоговое окно.

Выбор файла базы данных.

```
C:\Program Files\Borland\Delphi 3\Demos\Data\COUNTRY.DB
```

4. Нажмите кн. Next >.

Откроется диалоговое окно.

Выбор полей базы данных для размещения на форме в заданном порядке. Перенесите все поля из окна Available Fields в окно Ordered Select:

```
Available Fields    Ordered Select
Name                 Continent
Capital          →   Name
Continent            Capital
...                  ...
```

5. Нажмите кн. Next >.
Откроется диалоговое окно.
Положение полей на форме: Horizontally
6. Нажмите кн. Next >.
Откроется диалоговое окно.
Генерация программы: Generate a main form| Form Only.
Нажмите кн. Finish.
7. Удалите из проекта приложения форму, созданную по умолчанию: Project|Remove From Project...

```
Unit1  Form1
```

Нажмите кн. OK.
8. Все сохраните File|Save All.
9. Компилируйте и выполните приложение Run|Run.
10. Тестирование и отладка.
11. Вставьте в базу данных дополнительные описания:

```
Российская Федерация
Площадь      17075400
Население   143090000
...
Средний Урал
Площадь        194800
Население     4500000
...
```

2.5. Технология визуального проектирования в среде Delphi

Проектирование и выполнение приложения происходит с соблюдением определенной последовательности выполнения операций в интегрированной среде (Integrated Development Environment — IDE) визуального программирования Delphi (рис. 2.2).

1. Войдите в среду Delphi из меню операционной системы Windows.
Пуск| Все программы| Borland Delphi 7 | Delphi 7.

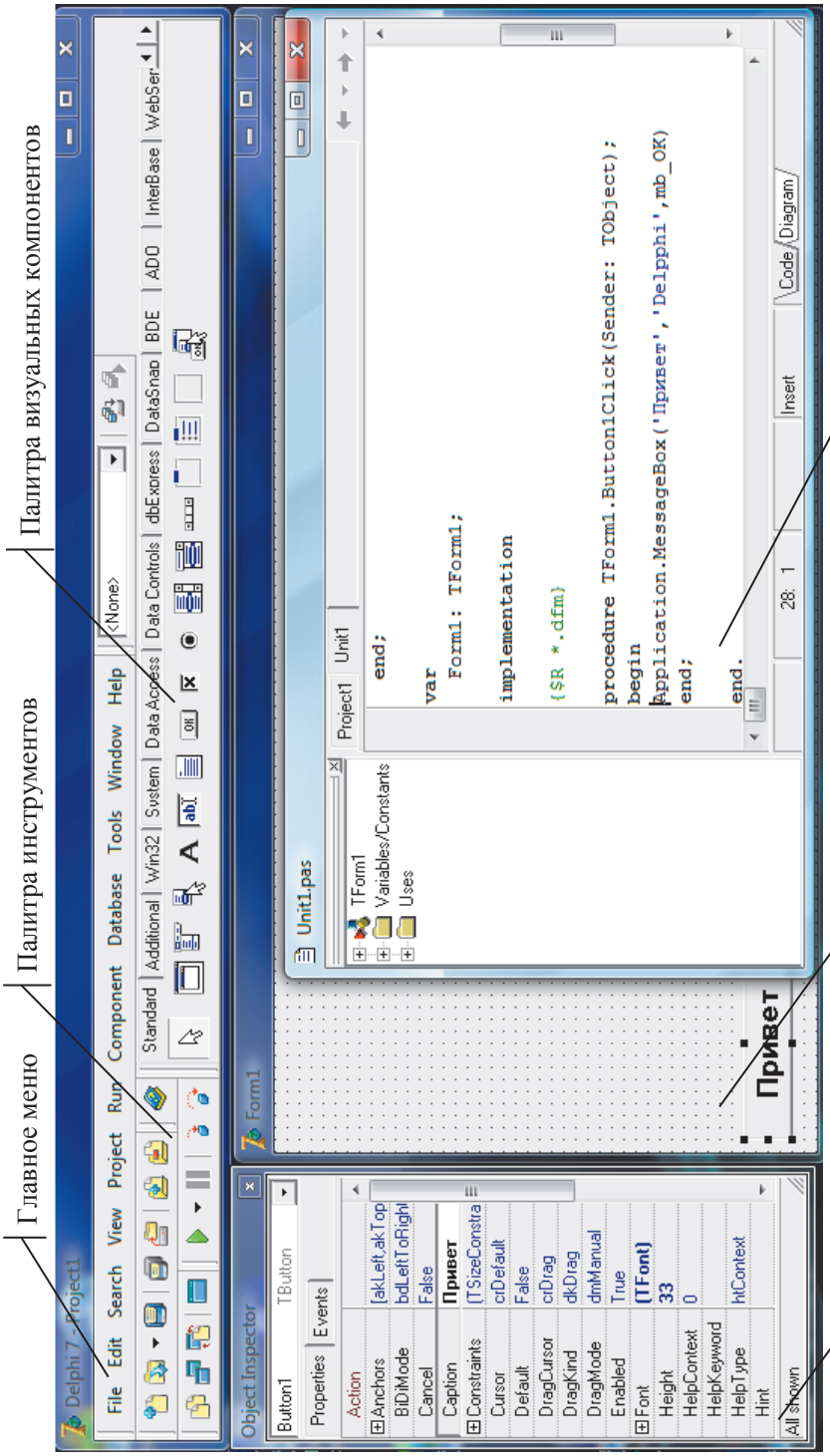


Рис. 2.2. Окна среды визуального программирования Delphi

2. Создайте шаблон нового проекта приложения командами главного меню.

File| New| Application.

3. Сохраните проект в отдельной папке, которую можно создать во время сохранения. В окнах диалогов сохранения файлов приложения укажите новые имена файлов.

File|Save Project As...

4. Перейдите в окно формы. Выбирайте на соответствующих страницах палитры необходимые визуальные компоненты и размещайте их на форме. Отрегулируйте размеры формы и компонентов. Установите взаимное положение компонентов на форме.

5. Выбирайте на форме по очереди визуальные компоненты. Переходите в инспектор объектов и на странице Properties настраивайте свойства выбранного компонента.

6. Сохраните файлы проекта приложения.

File|Save All.

Не забывайте выполнять эту очень важную операцию регулярно, каждые 15 минут работы над проектом, чтобы не потерять результаты своей работы.

7. Программируйте обработчики событий управляющих компонентов проекта приложения в редакторе кода. Если это событие по умолчанию, то переход в редактор происходит двойным щелчком левой кнопки мыши. Другие методы обработки программируются после выбора события компонента на странице Events в Инспекторе Объектов.

8. Сохраните файлы проекта приложения: File| Save All.

9. Запустите компиляцию и выполнение приложения: Run| Run.

10. Если обнаружены ошибки во время компиляции, то внизу окна редактора кода появятся сообщения об ошибках. Два раза щелкните по сообщению и вы попадете в место ошибки (локализуете). Ошибки следует локализовать и исправить, затем сохраните и выполните программу вновь (пункты 8 и 9).

11. Если обнаружена ошибка во время выполнения приложения, то произойдет исключение и появится диалоговое окно сообщения об ошибке. Для исправления ошибки в проекте нужно остановить отладку программы командами меню: Run| Program Reset.

12. Если приложение выполняется без ошибок, тестируйте все его функции. Можно в дальнейшем запускать исполняемый exe-файл приложения, который находится в папке проекта приложения.

Глава 3.
Обработка полей таблицы

Каждое поле (колонка) таблицы содержит данные одинакового типа. Совокупность полей определяет структуру таблицы БД.

3.1. Состояние набора данных

Состояние набора данных таблицы зависит от свойства State компонента Table. Возможны шесть состояний, которые становятся значениями State: dsEdit, dsBrowse, dsInsert, dsInactive, dsSetKey, dsCalcFields. Изменяют состояние методы Insert, Edit, Post, Cancel или Append. Проверку состояния можно осуществить во время выполнения программы через значение свойства State.

Если набор данных находится в режиме dsEdit, то изменения могут быть во многих полях записи. Метод Post сохранит изменения, а метод Cancel отменит изменения и восстановит начальные значения полей этой записи.

Подтверждение изменений в таблице контролируется в примере приложения с диалоговым окном на рис. 3.1.

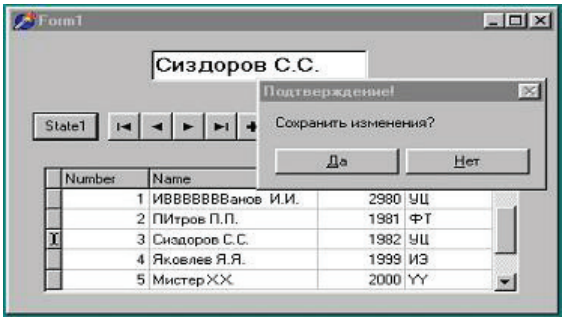


Рис. 3.1. Подтверждение изменений в таблице

Обработчик события OnClick кнопки State1 проверяет ответ диалога и в зависимости от ответа вызывает нужный метод — Post или Cancel.

```
procedure TForm1.State1Click (Sender: TObject);
var
  Save: Integer;
begin
  Save:= Application.MessageBox ('Сохранить изменения?',
        'Подтверждение!', mb_yesno);
  if Save = IDYES then
    studx.Post {Совершить изменения редактирования}
  else
    studx.Cancel; {Отменить изменения редактирования}
end;
```

Обратите внимание, что курсор в выбранной строке таблицы имеет вид вертикальной черты. Это обозначает режим просмотра.

3.2. Подтверждение изменений с проверкой состояния

Можно улучшить предыдущий проект, который не учитывает ситуацию, когда набор не находится в режиме dsEdit и выполняется метод Post или Cancel. Тогда возникает исключение о прерывании программы.

Усовершенствованный проект (рис. 3.2) проверяет значение состояния в свойстве State. Если это значение не равно dsEdit, то сообщается об ошибке.

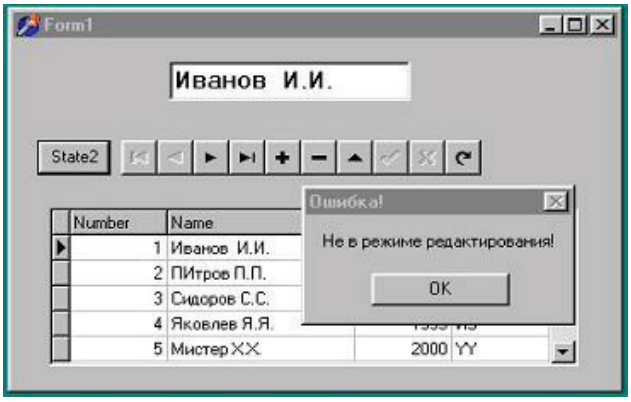


Рис. 3.2. Подтверждение изменений с проверкой состояния

Обработчик события OnClick кнопки State2 дополнен описанной проверкой.

```
procedure TForm1.State2Click (Sender: TObject);
var
  Save: Integer;
begin
if studx.State = dsEdit then
  begin
   Save:= Application.MessageBox ('Сохранить изменения?',
          'Подтверждение!', mb_yesno);
    if Save = IDYES then studx.Post {Учесть изменения}
    else
      studx.Cancel; {Отменить изменения редактирования}
  end
else
  Application.MessageBox ('Не в режиме редактирования!',
                          'Ошибка!', mb_ok);
end;
```

Обратите внимание, что курсор в выбранной строке таблицы имеет вид стрелки-треугольника, что обозначает режим редактирования. Примеры показали возможности управления записями при помощи программного кода.

3.3. Контроль данных. Метод Cancel

Пример контроля вводимых простых чисел (рис. 3.3). Форма содержит компоненты EntPrime (типа TDBEdit) и Primes (типа TTable).

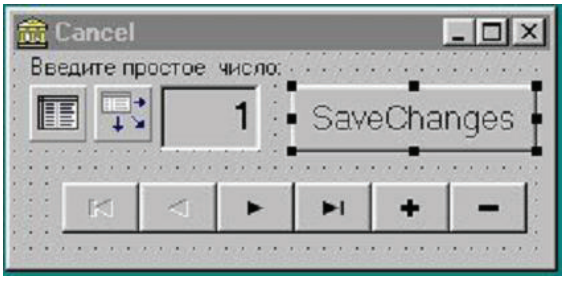


Рис. 3.3. Контроль данных. Метод Cancel

Обработчик события OnEnter объекта EntPrime методом Edit переводит таблицу Primes в состояние dsEdit:

```
procedure TForm1.EntPrimeEnter (Sender: TObject);
begin
  Nav.Enabled:=False; {Отключить навигатор}
  Primes.Edit; {Перевести таблицу в режим dsEdit}
end;
```

Обработчик события OnClick для кнопки Save (надпись SaveChanges) проверяет, простое число в рамке EntPrime или нет.

```
procedure TForm1.SaveClick (Sender: TObject);
var
  IsPrime: Boolean;       {True, если простое}
  Count: Integer;         {Делитель}
  NumToTest: Integer;     {Испытуемое число}
begin
  IsPrime:= True;
  Count:= 2;
  NumToTest:= StrToInt (EntPrime.Text);
  While (Count < NumToTest) and IsPrime do
    begin
      if NumToTest mod Count =0 then
        IsPrime:= False; {Обнаружено не простое число}
        Inc (Count);     {Увеличение делителя}
    end;{While}
  if IsPrime then         {Если число простое,}
    Primes.Post           {сохраняем его в базе данных,}
  else {иначе отмена, Cancel}
    begin
      Application.MessageBox ('Число не простое',
                              'Неверные данные', MB_OK);
      Primes.Cancel;
    end;{if}
  Nav.Enabled:= True;     {Включить навигатор}
end;
```

Таблица Primes привязана к БД, в которой будут храниться простые числа.

3.4. Доступ к полям в коде приложения

Свойство Fields таблицы набора данных описывает значения полей и структуру таблицы в виде массива имя_таблицы.Fields [индекс_поля]. Вот пример проверки имени и типа первого поля Number таблицы studx (рис. 3.4).

Обработчик события OnClick для кнопки Number выполняет эту проверку.

```
procedure TForm1.NumberClick (Sender: TObject);
begin
  if not (studx.Fields [0].DataType = ftAutoInc) then
    begin
      Application.MessageBox ('Ошибка в типе данных поля 0',
                              'Ошибка БД', mb_ok); exit;
    end; {if}
  if not (comparetext (studx.Fields [0].FieldName,'Number')=0) then
    begin
     Application.MessageBox ('Ошибка в имени поля 0.Ошибка БД',
       mb_ok);
     exit;
     end; {if}
   Application.MessageBox ('Поле 0 проверено: тип = ftAutoInc; '+
                           'имя = Number','Информация', mb_ok);
end;
```

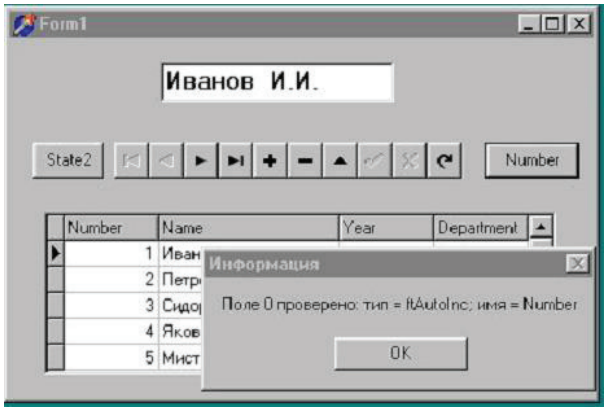


Рис. 3.4. Проверка имен и типов полей

Диалоговое окно показало положительные результаты проверки поля Number.

3.5. Заполнение полей данными

Delphi разрешает также вариант обращения к полю по имени в формате имя_таблицы.FieldByName ('имя_поля').тип_данных. Тип_данных определяется соответствующими свойствами:

```
AsBoolean: Boolean
AsDateTime: DateTime
AsFloat: Double
AsInteger: Integer
AsString: String
```

Этот вариант нагляднее, чем имя_таблицы.Fields [индекс_поля] в предыдущем примере. Свойство Fields также позволяет выбрать тип результата, написав Fields [индекс_поля].AsString.

3.5.1. Перемещение по записям

Мы перемещались по записям при помощи элементов управления. То же можно сделать в программе, используя код. После открытия таблицы следующим шагом нужно узнать, как перемещаться по записям внутри нее. Следующий обширный набор методов и свойств TDataSet обеспечивает все, что нужно для доступа к любой конкретной записи внутри таблицы:

```
procedure First;
procedure Last;
procedure Next;
procedure Prior;
property BOF: Boolean read FBOF;
property EOF: Boolean read FEOF;
procedure MoveBy (Distance: Integer);
```

Таким образом, компонент Table имеет пять методов движения по записям.

Метод	Действие
Next	Следующая
Prior	Предыдущая
First	Первая
Last	Последняя
MoveBy (I)	Переход на I позиций

Переход на новую запись неявно вызывает метод Post и изменения в записи сохраняются. После этого нельзя отменить изменения, вызвав метод Cancel.

Часто необходимо знать о достижении начала или конца набора данных. Для чего служат свойства BOF и EOF. В начале таблицы BOF = True. В конце таблицы EOF = True.

Дадим краткий обзор их функциональных возможностей:

- вызов Table1.First перемещает вас к первой записи в таблице;
- Table1.Last перемещает вас к последней записи;
- Table1.Next перемещает вас на одну запись вперед;
- Table1.Prior перемещает вас на одну запись назад;
- можно проверять свойства BOF или EOF, чтобы понять, находитесь ли вы в начале или в конце таблицы;
- процедура MoveBy перемещает вас на I записей вперед или назад в таблице. Нет никакого функционального различия между запросом Table1.Next и вызовом Table1.MoveBy (1). Аналогично вызов Table1.Prior имеет тот же самый результат, что и вызов Table1.MoveBy (-1).

3.5.2. Заполнение полей данными

Для того чтобы начать использовать эти навигационные методы, нужно поместить TTable, TDataSource и TDBGrid на форму. Присоедините DBGrid1 к DataSource1 и DataSource1 к Table1. Затем установите свойства таблицы:

- в DatabaseName — имя подкаталога, где находится таблица studx.db;
- в TableName установите имя таблицы studx.

Рассмотрим пример приложения, заполняющего два поля таблицы (Object, Mark) строковыми и целочисленными данными (рис. 3.5).

Обработчик события OnClick для кнопки Mark заполняет поля Object и Mark указанными значениями.

```
procedure TForm1.MarkClick (Sender: TObject);
begin
   studx.First;            {К началу таблицы}
   studx.DisableControls; {Ускоряет работу отключением управления}
   while not (studx.EOF) do {Проверить конец таблицы}
     begin
```

```
        studx.Edit;
        {Задать название дисциплины в поле с индексом 4}
        studx.Fields [4].AsString:= 'Информатика';
        {Получить случайное число, присвоить полю с индексом 5}
       studx.Fields [5].AsInteger:= random (5)+1;//от двух до пяти
        studx.Post; {В действительности не требуется}
        studx.Next; {К следующей записи}
     end; {while}
   studx.EnableControls; {Восстанавливает обработку}
end; {procedure}
```

Органы управления studx.DisableControls отключены для ускорения работы приложения на время выполнения цикла.

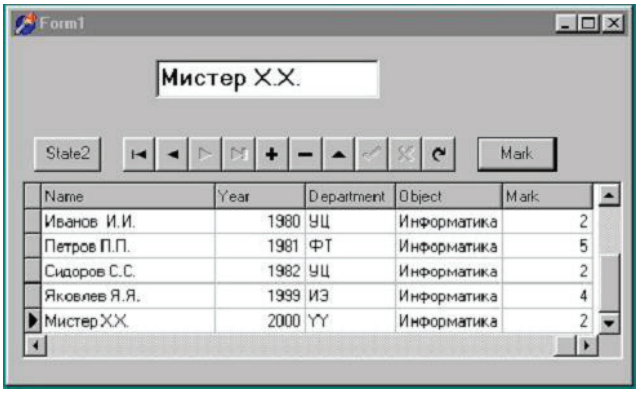


Рис. 3.5. Заполнение полей данными в приложении

TDataSet.BOF — это read-only Boolean свойство, которое используется для проверки, находится ли указатель в начале таблицы. Свойства BOF возвращает true в трех случаях:

- после того как вы открыли файл;
- после того как вы вызывали TDataSet.First;
- после того как вызов TDataSet.Prior не выполняется.

Первые два пункта — очевидны. Когда открывается таблица, Delphi помещает указатель на первую запись; когда вызывается метод First, Delphi также перемещает указатель в начало таблицы. Третий пункт, однако, требует небольшого пояснения: после того как вызыван метод Prior много раз, можно добраться до начала таблицы, и следующий вызов Prior будет неудачным — после этого BOF и будет возвращать True.

Следующий код показывает самый общий пример использования Prior, когда вы попадаете к началу файла:

```
while not Table.Bof do begin
  DoSomething;
  Table1.Prior;
end;
```

В коде, показанном здесь, гипотетическая функция DoSomething будет вызвана вначале на текущей записи и затем на каждой следующей записи (от текущей и до начала таблицы). Цикл будет продолжаться до тех пор, пока вызов Table1.Prior не сможет больше переместить указатель на предыдущую запись в таблице. В этот момент BOF вернет True и программа выйдет из цикла. Для того чтобы оптимизировать приведенный код, установите в False значение DataSource1.Enabled перед циклом и верните ему True после окончания цикла.

Все сказанное относительно BOF также применимо и к EOF. Другими словами, код, приведенный ниже, показывает простой способ пробежать по всем записям в dataset:

```
Table1.First;
while not Table1.EOF do begin
  DoSomething;
  Table1.Next;
end;
```

Классическая ошибка в случаях, подобных этому: вход в цикл while или repeat, без вызова Table1.Next:

```
Table1.First;
repeat
  DoSomething;
until Table1.EOF;
```

Если случайно написан такой код, то машина зависнет, и можно выйти из цикла только нажав Ctrl-Alt-Del и прервав текущий процесс. Также этот код мог бы вызвать проблемы, если вы открыли пустую таблицу. Так как здесь используется цикл **repeat**, DoSomething был бы вызван один раз, даже если бы нечего было обрабатывать. Поэтому лучше использовать цикл **while** вместо **repeat** в ситуациях, подобных этой.

EOF возвращает True в следующих трех случаях:

- после того как вы открыли пустой файл;
- после того как вы вызывали TDataSet.Last;
- после того как вызов TDataSet.Next не выполняется.

3.6. Заполнение полей вычисленными данными

Если требуется получить новые данные из БД, но не хранить их в БД, Delphi определяет дополнительные вычисляемые поля (Calculated Fields, рис. 3.6).

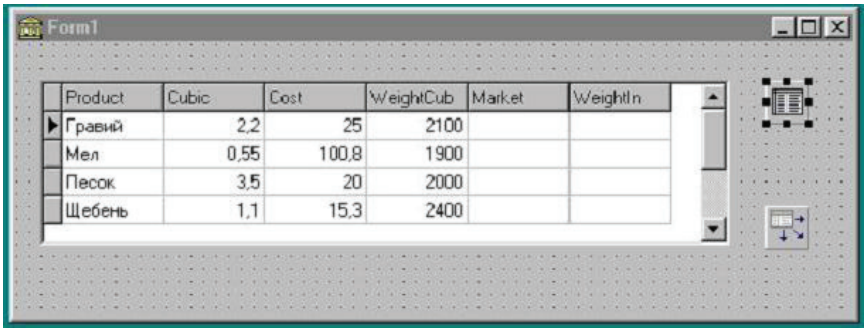


Рис. 3.6. Форма для заполнения полей вычисленными данными

В локальном меню РПТ таблицы выполняются команды Add Fields...| New Field... В диалоговом окне задается имя поля и его вид Calculated.

Например, добавим в таблицу два поля для вычисления произведений: Market = Cubic*Cost и WeightIn = Cubic*WeightCub. Окно выполненного приложения представлено на рис. 3.7.

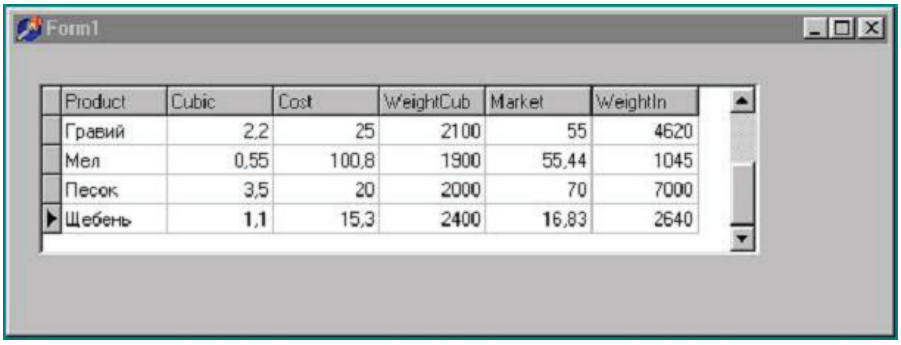


Рис. 3.7. Окно выполненного приложения

Вычисляемые поля существуют только во время работы приложения.

Обработчик события OnCalcFields компонента Invent (типа TTable) создается в инспекторе объектов на странице Events. Нужно два раза

щелкнуть левой кнопкой мыши справа от имени события OnCalcFields, и откроется окно редактора кода, в котором нужно вписать код метода обработки события.

```
procedure TForm1.InventCalcFields (DataSet: TDataSet);
var
  CurrentPrice: Double;
  WeightCub: Double;
  CubicIn: Double;
begin
  {Представим значения полей локальными переменными}
  CurrentPrice:= Invent.FieldByName ('Cost').AsFloat;
  WeightCub:= Invent.FieldByName ('WeightCub').AsFloat;
  CubicIn:= Invent.FieldByName ('Cubic').AsFloat;
  {Присвоим значения вычисляемым полям}
 Invent.FieldByName ('Market').AsFloat:= CurrentPrice*CubicIn;
  Invent.FieldByName ('WeightIn').AsFloat:= WeightCub*CubicIn;
end;
```

Глава 4.
Обработка записей

Изучим методы работы со строками таблицы: упорядочивание, поиск, выбор диапазона, фильтрация.

4.1. Упорядочивание записей

4.1.1. Индексы для упорядочивания записей таблицы

Индексы полей определяют сортировку записей в таблице. В СУБД Paradox и dBase при определении ключевого поля автоматически возникает индекс этого поля. Главное достоинство индексации состоит в том, что становится возможным и ускоряется поиск по индексированным полям.

Возможна многократная индексация таблиц. Рост производительности сопровождается недостатками: замедляется модификация, ввод-вывод и передача данных, возрастает размер таблиц. Поэтому нужно индексировать только часто сортируемые столбцы.

Задание 1. Индексируйте поля Name и Number в таблице studx.db

1. Откроем DBD: Tools| Database Desktop.

2. Выберите индексируемую БД: File| Open| Table...

Примечание. Если в проекте Delphi открыта таблица, закройте ее, установив в свойстве Active значение False.

3. Индексация — это изменение структуры таблицы БД.

Table| Restructure...

4. Появится диалоговое окно со списком полей Field Roster. В списке Table Properties выберите свойство Secondary Indexis.

5. Нажмите кн. Define... Появится диалоговое окно определения индексов. Выбирайте в левой части окна Fields по очереди поля и ко-

пируйте их с помощью стрелки в правую часть Indexed fields. Например, выберем поле Name.

6. Нажмите кн. OK. В рамке свойств Index options установите флажок Maintained. Определите в рамке Index name имя индекса NameInd. Повторите шаги индексации для поля Number.

7. Сохраните изменения структуры. Нажмите кн. Save.

4.1.2. Проект упорядочивания таблицы

Свойство IndexName компонента Ttable определяет, какой индекс будет использоваться при упорядочивании как текущий. Если ни один из индексов не определен, то для сортировки будет использован ключ — первичный индекс.

Другой способ заключается в использовании свойства IndexFieldNames, которое содержит имя поля. Эти два свойства нельзя устанавливать одновременно. Если задано одно, Delphi автоматически очищает другое.

Задание 2. Меню упорядочивания таблицы по выбранным полям.

1. Добавьте компоненты для работы с таблицей на форму проекта приложения и настройте их (рис. 4.1).

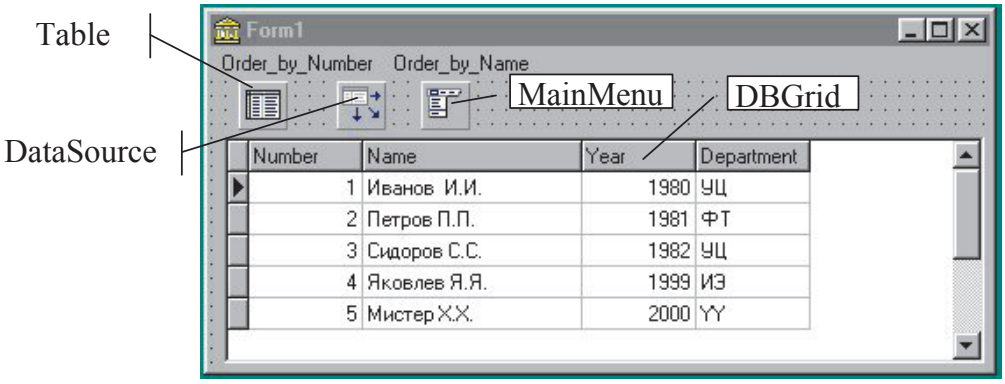


Рис. 4.1. Форма содержит два пункта меню сортировки

2. Форма должна содержать два пункта меню сортировки: по номеру и по фамилии. Добавим главное меню с двумя разделами Order_by_Number и Order_by_Name. Из панели компонентов со страницы Standard поместите на форму компонент MainMenu.

3. В инспекторе объектов выберите свойство Items и в нем задайте два элемента со свойствами — у одного Name = Order_by_Number, а у другого Name = Order_by_Name. Закройте окно определений элементов ☒.

На форме появятся два заданных пункта меню: Order_by_Number и Order_by_Name.

Задание 3. Задайте обработчики событий OnClick пунктов меню.

1. Щелчок по первому пункту меню вызовет окно редактора кода модуля в позиции обработчика OnClick выбора этого пункта. Введите соответствующие коды по первому заголовку меню:

```
procedure TForm1.Order_by_NumberClick (Sender: TObject);
begin
  Order_by_Name.Checked:= False;
  Order_By_Number.Checked:= True;
  studx.IndexFieldNames:= 'Number';
end;
```

2. Введите метод обработки события OnClick по второму заголовку меню Order_by_NameClick:

```
procedure TForm1.Order_by_NameClick (Sender: TObject);
begin
  Order_By_Number.Checked:= False;
  Order_by_Name.Checked:= True;
  studx.IndexFieldNames:= 'Name';
end;
```

Свойство Checked переключает активность команды меню. Пассивная команда окрашена серым цветом и не доступна для выполнения.

Задание 4. Тестируйте и отладьте проект.

1. Выполните приложение Run| Run.

2. После компиляции и выполнения приложения появится окно с двумя командами главного меню, которые позволяют упорядочить записи таблицы по именам или по номерам. По очереди выполняйте пункты меню и проверяйте результаты сортировки. Окно приложения после сортировки по имени показано на рис. 4.2. Произошло упорядочивание имен по алфавиту.

Индекс упорядочивания назначается обработчиком события команды меню в свойстве IndexFieldNames. Если выполнить команду меню Order_By_Number, восстановится начальный вид таблицы, упорядоченный по номеру записи.

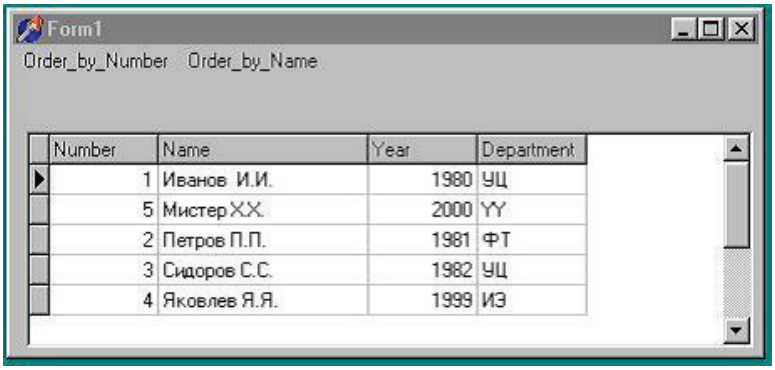


Рис. 4.2. Окно приложения после сортировки по имени

4.2. Поиск записей в таблице

Средствами компонента Ttable можно организовать мощные процедуры поиска. Возможен поиск только по индексированным полям. Есть две методики поиска записей: SetKey, FindKey.

4.2.1. Метод SetKey

1. Методом studx.SetKey (для таблицы studx) таблица переводится в состояние State = dsSetKey.

```
Studx.SetKey;
```

2. Теперь установим индекное поле поиска равным искомому значению. В коде это выглядит как установление значения соответствующего поля текущей записи, но состояние dsSetKey вызовет поиск, а не редактирование (dsEdit). Например, поиск первичного ключа номер три:

```
Studx.FieldByName ('Number').AsInteger:= 3;
```

3. Завершим переходом к искомой записи.

```
Studx.GoToKey;
```

Если запись не найдена, метод вернет значение False. Метод работает, если известно точное искомое значение поля, тогда результат будет True.

4. Если известна только часть поля или приблизительное значение, используется метод GoToNearest.

```
Studx.GoToNearest;
```

Метод отыскивает лучшее совпадение или просматривает таблицу до конца.

Задание 1. Организуйте поиск по ближайшему первичному ключу.

1. Поместите на форму компоненты: Studx: TTable, DBGrid1: TDBGrid, DataSource1: TDataSource и настройте их свойства. Окно редактирования teNumberSearch: TEdit нужно для ввода номера искомой записи.

2. Добавьте на форму кн. SearchNumber с обработкой события OnClick:

```
procedure TSearchN1.SearchNumberClick (Sender: TObject);{Вариант 1}
begin Studx.SetKey;
  Studx.FieldByName ('Number').AsInteger:=
        StrToInt (teNumberSearch.text); Studx.GoToNearest;
end;
```

3. Приложение будет устанавливать в качестве текущей запись с номером, ближайшим к значению, введенному в окне редактирования teNumberSearch.

Задание 2. Короткий вариант поиска по первичному ключу

Есть два сокращенных способа поиска по ключу: FindKey, FindNearest. Их выполнение инкапсулирует в одной функции SetKey установку критерия поиска и переход.

Метод FindNearest заменяет три оператора предыдущего обработчика.

```
procedure TSearchN2.SearchNumberClick (Sender: TObject);{Вариант 2}
begin Studx.FindNearest ([teNumberSearch.text]) end;
```

Задание 3. Поиск по вторичным индексам.

1. На новой форме окно ввода называется teNameSearch: Tedit.

2. Измените также кнопку поиска по имени SearchName с обработкой события OnClick:

```
procedure TSearchN3.SearchNameClick (Sender: TObject);
begin
  Studx.IndexName:= 'NameInd';
  Studx.EditKey;
  Studx.FieldByName ('Name').AsString:= teNameSearch.text;
  Studx.GoToNearest;
end;
```

3. Окно приложения с результатами поиска ближайшего имени показано на рис. 4.3.

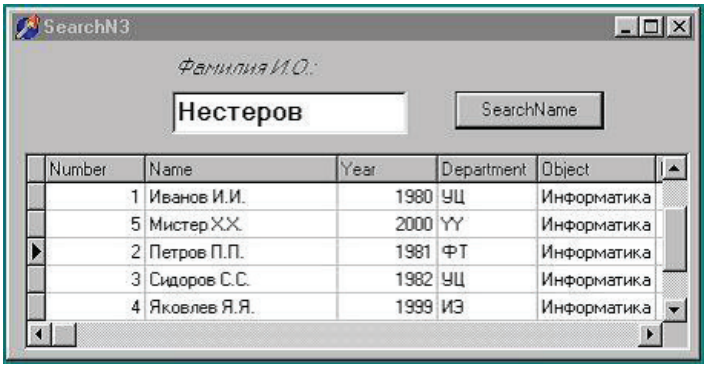


Рис. 4.3. Окно приложения с результатами поиска ближайшего имени

4.2.2. Основные понятия о TDataSource

Класс TDataSource используется в качестве проводника между TTable или TQuery и компонентами, визуализирующими данные, типа TDBGrid, TDBEdit и TDBComboBox (data-aware components). В большинстве случаев все, что нужно сделать с DataSource, — это указать в свойстве DataSet соответствующий компонент TTable или TQuery. Затем у компонента презентации в свойстве DataSource указывается имя компонента TDataSource, который используется в настоящее время.

TDataSource также имеет свойство Enabled, и оно может быть полезно всякий раз, когда вы хотите временно отсоединить, например, DBGrid от таблицы или запроса. Это требуется, к примеру, если нужно программно пройти через все записи в таблице. Ведь если таблица связана с визуальными компонентами (DBGrid, DBEdit и т. п.), то каждый раз, когда вы вызываете метод TTable.Next, визуальные компоненты будут перерисовываться. Даже если само сканирование в таблице двух или трех тысяч записей не займет много времени, то может потребоваться значительно больше времени, для того чтобы столько же раз перерисовать визуальные компоненты. В случаях, подобных этому, лучше всего установить поле DataSource.Eabled в False. Это позволит вам просканировать записи без перерисовки визуальных компонентов. Это единственная операция, которая может увеличить скорость в некоторых случаях на несколько тысяч процентов.

4.3. Выбор диапазона записей

Компонент таблицы может работать с подмножеством записей в таблице базы данных, используя диапазоны, удовлетворяющие заданным условиям отбора. Например, найти записи, содержащие дисциплину ‘Информатика’, или выбрать всех студентов с оценками 4 или 5. После отбора можно считать, что остальные записи просто не существуют.

Используем метод набора данных SetRange для установки критериев отбора. Отменяет ограничение диапазона записей метод CancelRange.

Задание 1. Организуйте выбор диапазона записей в БД studx.db по дисциплине или по оценке Mark.

1. Необходимо предварительно индексировать два поля средствами Database Desktop: поле Object индексировать с именем ObjectInd, поле Mark индексировать с именем MarkInd.

2. Добавьте на форму три кнопки: RangeObj, RangeMark и All.

3. Первая кнопка RangeObj имеет обработчик:

```
procedure TRange.RangeObjClick (Sender: TObject);
begin
  Studx.IndexName:= 'ObjectInd';
  Studx.SetRange (['Енформатика'], ['Информатика']);
end;
```

Диапазон выделяется между двумя словами — ‘Енформатика’ и ‘Информатика’.

4. Вторая кнопка RangeMark имеет обработчик:

```
procedure TRange.RangeMarkClick (Sender: TObject);
begin
  Studx.IndexName:= 'MarkInd';
  Studx.SetRange ([4], [5]);
end;
```

Выделяется диапазон оценок от 4 до 5.

5. Третья кнопка All имеет обработчик:

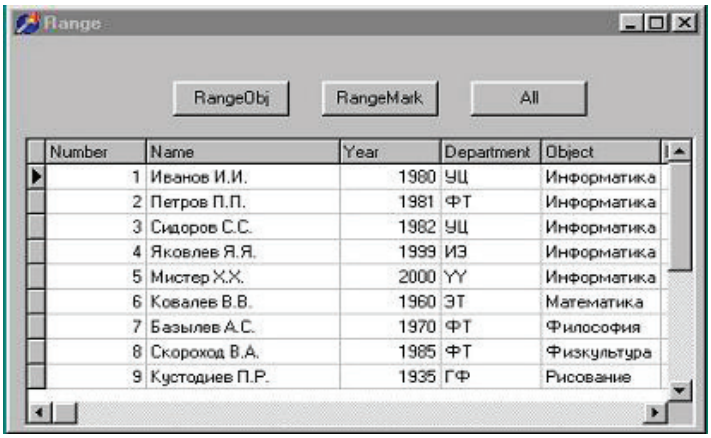
```
procedure TRange.AllClick (Sender: TObject);
begin  Studx.CancelRange;  end;
```

Отменяются диапазоны и становятся доступны все записи с порядком, заданным последним назначенным индексом.

Задание 2. Тестируйте и отладьте проект.

1. Выполните приложение: Run| Run.

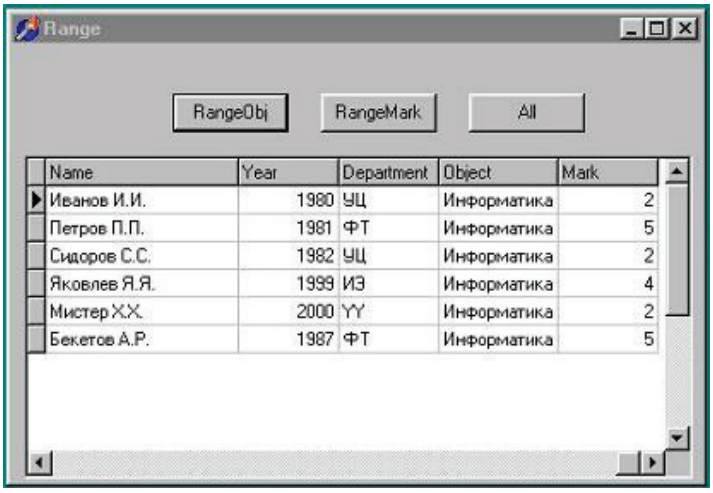
2. Окно приложения в начальном состоянии будет содержать все записи таблицы, упорядоченные по номеру (рис. 4.4).



Number	Name	Year	Department	Object
1	Иванов И.И.	1980	УЦ	Информатика
2	Петров П.П.	1981	ФТ	Информатика
3	Сидоров С.С.	1982	УЦ	Информатика
4	Яковлев Я.Я.	1999	ИЭ	Информатика
5	Мистер Х.Х.	2000	YY	Информатика
6	Ковалев В.В.	1960	ЭТ	Математика
7	Базылев А.С.	1970	ФТ	Философия
8	Скороход В.А.	1985	ФТ	Физкультура
9	Кустодиев П.Р.	1935	ГФ	Рисование

Рис. 4.4. Окно приложения в начальном состоянии

3. Окно приложения после нажатия кнопки RangeObj будет содержать записи только по предмету ‘Информатика’ (рис. 4.5).



Name	Year	Department	Object	Mark
Иванов И.И.	1980	УЦ	Информатика	2
Петров П.П.	1981	ФТ	Информатика	5
Сидоров С.С.	1982	УЦ	Информатика	2
Яковлев Я.Я.	1999	ИЭ	Информатика	4
Мистер Х.Х.	2000	YY	Информатика	2
Бекетов А.Р.	1987	ФТ	Информатика	5

Рис. 4.5. Окно приложения с результатами отбора предмета

4. Если нажать кнопку RangeMark, окно покажет оценки 4 и 5 (рис. 4.6).

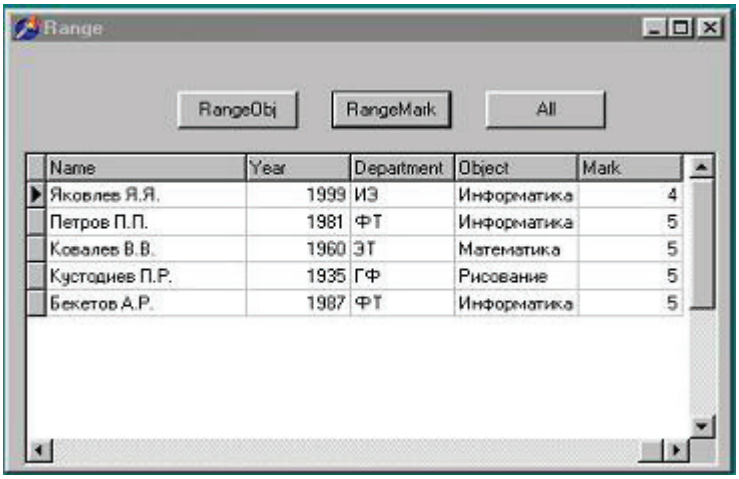


Рис. 4.6. Окно приложения с результатами отбора оценок

5. Окно после отмены выбора кн. All покажет все записи (рис. 4.7).

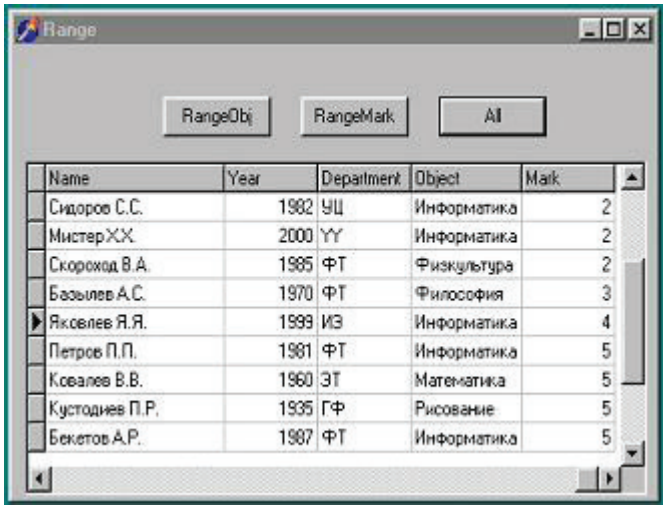


Рис. 4.7. Окно приложения после отмены выбора диапазонов

Упорядочивание строк таблицы осталось по последнему выбранному индексу оценки MarkInd.

Задание 3. Допишите в методе TRange.AllClick один оператор, восстанавливающий начальный порядок строк таблицы по номеру.

4.4. Фильтры

4.4.1. Свойства фильтрации компонента TTable

Компонент таблицы может также работать с подмножеством записей в таблице базы данных, используя фильтры.

TBDEDataSet.Filter

Определяет текст текущего фильтра Filter: string для набора данных.

Описание

Filter используется, для того чтобы определить фильтр набора данных. Когда фильтрация применяется к набору данных, только записи, которые подходят условиям фильтра, становятся доступными для приложения. Фильтр содержит строку, которая описывает условие фильтра. Например, следующее условие фильтра отображает только те записи, где поле State — 'CA' или 'MA':

```
State = 'CA' or State = 'MA'
```

Чтобы фильтровать строки на основе частичных сопоставлений, используйте звездочку как знак подстановки. Например: `State = 'M*'`

Примечание. Приложения могут менять Filter во времени выполнения программы, изменяя условие фильтрации для набора данных (например, как ответ на ввод пользователя).

TBDEDataSet.Filtered

Свойство Filtered определяет, активна ли фильтрация для набора данных.

```
property Filtered: Boolean;
```

Описание

Проверьте Filtered, чтобы определить, действует ли фильтр набора данных. Если Filtered имеет значение True, то фильтрация активна. Чтобы применять условия фильтра, определенные в свойстве Filter или обработчике события OnFilterRecord, установите Filtered равным True.

Примечание. Когда назначена фильтрация, редактирование пользователем записи может привести к тому, что запись больше не соответствует условию фильтра. В следующий раз, когда запись снова ищется в наборе данных при действующем фильтре, может казаться, что запись исчезла. Если это случается, то следующая запись, которая подходит под условие фильтра, становится текущей записью.

TBDEDataSet.FilterOptions

Определяет, чувствительна или нет фильтрация к регистру и разрешаются или нет частичные сравнения при фильтрации записей.

```
property FilterOptions: TFilterOptions;
```

Описание

Установите FilterOptions, чтобы определить, чувствительна или нет фильтрация к регистру букв при фильтрации строковых или символьных полей и допускаются или нет частичные сопоставления при соответствии условиям фильтра.

По умолчанию в FilterOptions установлено пустое множество. Для фильтров строковых полей включите foCaseInsensitive в FilterOptions, чтобы охватить все изменения строки независимо от перехода в верхний регистр.

Для того чтобы предотвратить частичные строковые сравнения, включите в FilterOptions значение foNoPartialCompare.

Примечание. Чтобы фильтровать строки на основе частичных сравнений, исключите foNoPartialCompare из FilterOptions и используйте звездочку как групповой символ. Например: `State = 'M*'`

Типы: TFilterOption, TFilterOptions

Влияют на то, как применяются строки фильтра.

```
Unit db;
type
  TFilterOption = (foCaseInsensitive, foNoPartialCompare);
  TFilterOptions = set of TFilterOption;
```

Описание

TfilterOptions: это множество, которое определяет, как выражения фильтра применяются к строковым полям. Оно может содержать нуль или большее количество значений: foCaseInsensitive, foNoPartialCompare.

Смысл значений

foCaseInsensitive: литеральные строки в фильтре сравниваются со строковыми полями без учета регистра.

foNoPartialCompare: звездочки (*) в фильтре обрабатываются как групповые символы. Когда в настройках фильтра foNoPartialCompare не включено, строки, которые заканчиваются звездочкой, выражают частичное соответствие со звездочкой, соответствующей любому числу символов.

4.4.2. Форма проекта фильтрации

Задание 1. Разместите компонеты проекта приложения на форме (рис. 4.8).

Добавьте через DBD в таблицу БД study.db индексы: поля Object — индекс упорядочивания по дисциплине ObjectInd, поля Year — индекс упорядочивания по году YearInd, двух полей Object и Year — индекс упорядочивания по дисциплинам, а внутри одной дисциплины по году ObjYearInd.

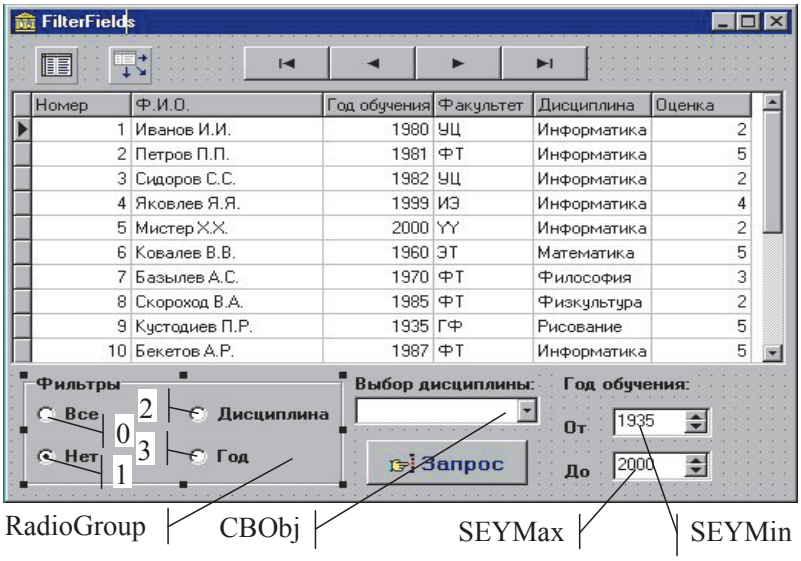


Рис. 4.8. Фильтры

Последовательность кнопок в радиогруппе RadioGroup1 определяется свойством ItemIndex. Для кнопки «Все» ItemIndex = 0, для кнопки «Нет» ItemIndex = 1, для кнопки «Дисциплина» ItemIndex = 2, для кнопки «Год» ItemIndex = 3.

Компоненты SEYMax и SEYMin определяют границы лет. Они состоят из двух типов объектов TEdit и UpDown, которые объединяются свойством UpDown.Associate.

Свойства основных объектов на форме

```
object Form1: TForm1
  Caption = 'FilterFields'
  Color = clBtnFace
```

```
object SEYMax: TSpinEdit
  MaxValue = 2000
  MinValue = 1935
```

```
Font.Charset = DEFAULT_CHARSET
Font.Color = clWindowText
Font.Height = -11
Font.Name = 'MS Sans Serif'
Font.Style = []
OldCreateOrder = False
OnCreate = FormCreate
OnDestroy = FormDestroy
PixelsPerInch = 96
TextHeight = 13
object RadioGroup1: TradioGroup
  Left = 8
  Top = 248
  Width = 209
  Height = 81
  Caption = 'Фильтры'
  Columns = 2
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -12
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  ItemIndex = 1
  Items.Strings = (
    'Все'
    'Нет'
    'Дисциплина'
    'Год')
  ParentFont = False
  TabOrder = 2
  OnClick = RadioGroup1Click
end
object CBObj: TcomboBox
  Left = 224
  Top = 264
  Width = 121
  Height = 21
  ItemHeight = 13
  TabOrder = 5
  Value = 2000
end
  object BitBtn1: TBitBtn
  Left = 232
  Top = 296
  Width = 105
  Height = 33
  Caption = 'Запрос'
 Font.Charset=DEFAULT_CHARSET
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  ParentFont = False
  TabOrder = 6
  OnClick = RadioGroup1Click
  Glyph.Data = {find.bmp}
  NumGlyphs = 2
end
  object Table1: TTable
    Active = True
    DatabaseName = 'MyDat'
  TableName = 'study.DB'
  Left = 16
  Top = 8
  object Table1Number:
    DisplayLabel = 'Номер'
    FieldName = 'Number'
    ReadOnly = True
end
  object Table1Name:
  DisplayLabel = 'Ф.И.О.'
  FieldName = 'Name'
end
  object Table1Year:
 DisplayLabel = 'Год обучения'
  FieldName = 'Year'
end
```

```
    Items.Strings = (
      'Информатика'
      'Рисование'
      'Физкультура'
      'Философия'
      ' ')
    TabOrder = 3
  end
  object SEYMin: TSpinEdit
    Left = 392
    Top = 272
    Width = 65
    Height = 22
    MaxValue = 2000
    MinValue = 1935
    TabOrder = 4
    Value = 1935
  end
    object Table1Department:
      DisplayLabel = 'Факультет'
      FieldName = 'Department'
      Size = 10
    end
    object Table1Object:
      DisplayLabel = 'Дисциплина'
      FieldName = 'Object'
      Size = 12
    end
    object Table1Mark:
      DisplayLabel = 'Оценка'
      FieldName = 'Mark'
    end
  end
    object DataSource1:
      DataSet = Table1
      Left = 64
  end
```

Задание 2. Введите коды обработки событий для элементов управления.

1. Метод обработки события OnClick группы радиокнопок RadioGroup1:

```
procedure TForm1.RadioGroup1Click (Sender: TObject);
begin
  Table1.IndexFieldNames:='Number';
  if (RadioGroup1.ItemIndex = 1) then Table1.Filtered:= False
   else
    begin
     if (RadioGroup1.ItemIndex = 2) then
      begin
      Table1.Filter:= 'Object=''' +CBObj.Text+ '''';
      Table1.IndexName:='ObjectInd';
      end
     else if (RadioGroup1.ItemIndex = 3) then
    begin
    Table1.Filter:= ' ('+IntToStr (SEYMin.Value)+
     '<=Year)and (Year<='+IntToStr (SEYMax.Value)+')';
```

```
      Table1.IndexName:='YearInd';
      end
    else
      begin
      Table1.Filter:= ' (Object=''' +CBObj.Text+ ''')and ('
        +IntToStr (SEYMin.Value)+ '<=Year)and (Year<='
        +IntToStr (SEYMax.Value)+')';
      Table1.IndexName:='ObjYearInd';
      end;
  Table1.Filtered:= True;
  end;
end;
```

2. Метод обработки события OnFormCreate формы с инициализацией значений:

```
procedure TForm1.FormCreate (Sender: TObject);
begin Table1.Active:= True; CBObj.ItemIndex:= 0 end;
```

3. Завершающий метод обработки события OnFormDestroy формы с закрытием БД:

```
procedure TForm1.FormDestroy (Sender: TObject);
begin Table1.Active:= False; end;
```

4. Метод обработки события OnClick кнопки с надписью «Запрос» совпадает с обработчиком RadioGroup1Click.

Задание 3. Тестируйте и отладьте проект.

1. Выполните приложение: Run| Run.

2. После компиляции и выполнения приложения появится окно приложения. По очереди выбирайте настройки фильтров и элементами управления проверяйте результаты фильтрации.

Глава 5.
Таблицы и файлы

5.1. Создание таблиц в программе

Таблицу базы данных можно создать разными способами:

- в DBD;
- в среде Delphi;
- в приложении.

Рассмотрим следующий способ. Нужно создать в приложении экземпляр объекта типа TTable. Компонент таблицы имеет метод CreateTable, который создает новый файл БД во время выполнения программы.

Перед вызовом метода CreateTable необходимо установить значения свойств:

```
TableType       - тип таблицы,
DatabaseName    - база данных,
TableName       - имя таблицы,
FieldDefs       - массив описаний полей,
IndexDefs       - массив описаний индексов.
```

Свойство TableType имеет тип TTableType и определяет структуру БД, которой принадлежит таблица.

Описание

```
type TTableType = (ttDefault, ttParadox, ttDBase, ttASCII, ttFoxPro);
property TableType: TTableType;
```

Применяют TableType для того, чтобы описать структуру таблицы, принадлежащей ДБ типа dBASE, Paradox, FoxPro или ASCII. TableType может принимать следующие значения:

```
Значение Описание
ttDefault Тип на основании расширеня файла ДБ.
ttParadox Таблица имеет тип Paradox.
ttDBase   Таблица имеет тип dBASE.
ttFoxPro  Таблица имеет тип FoxPro.
ttASCII   Таблица имеет текстовый тип.
```

Если свойство TableType установлено в ttDefault, тип таблицы определяется по расширению файла, содержащего эту таблицу:

- расширение.DB или без расширения: таблица Paradox;
- расширение.DBF: таблица dBASE;
- расширение.TXT: таблица ASCII (текстовый файл).

5.1.1. Сценарий проекта приложения

1. Опишите и установите в программе компонент TTable.
2. Опишите и задайте свойства TTable: DatabaseName, TableName.
3. Установите тип таблицы TableType:= ttDefault или другой тип.
4. Уничтожьте все существующие определения полей таблицы Clear.
5. Добавьте новые определения полей в свойстве FieldDefs методом Add.
6. Очистите существующие индексные определения.
7. Добавьте описание индексного поля.
8. Вызовите метод CreateTable.

5.1.2. Форма проекта приложения

Разместите компонеты на форме проекта (рис. 5.1).

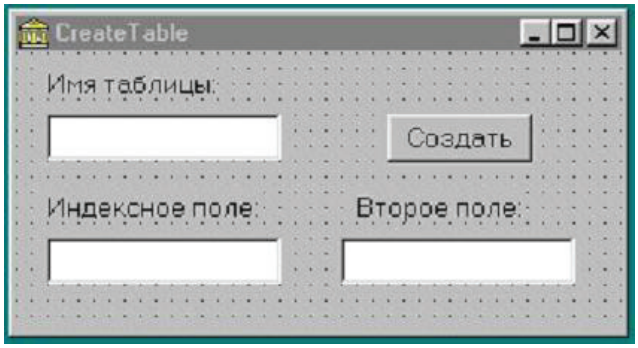


Рис. 5.1. Форма для создания таблицы

5.1.3. Обработчик события OnClick кнопки ButCreate (Создать)

```
procedure TCreateTable.ButCreateClick (Sender: TObject);
var NewTable: TTable; NewIndexOptions: TIndexOptions;
begin
  NewTable:= TTable.Create (Form1);
  NewIndexOptions:= [ixPrimary, ixUnique];
  with NewTable do begin
    Active:= False; DatabaseName:= '.\';
    TableName:= Edit1.Text; TableType:= ttDefault;
    FieldDefs.Clear;
    FieldDefs.Add (Edit2.Text, ftInteger, 0, False);
    FieldDefs.Add (Edit3.Text, ftInteger, 0, False);
    IndexDefs.Clear;
   IndexDefs.Add ('PrimaryIndex', Edit2.Text, NewIndexOptions);
  end; {Возможная проверка таблицы на существование}
    NewTable.CreateTable; {создать таблицу}
end;
```

Для того чтобы удалить таблицу из базы данных, вызовите метод DeleteTable компонента TTable. Например, следующий код удаляет таблицу БД, связанную с CustomersTable:

```
CustomersTable.DeleteTable;
```

5.2. Диалоги открытия и сохранения файлов

Есть возможность открывать любые файлы и работать с файлами по нашему выбору. Естественно, Delphi предоставляет такую возможность. Рассмотрим компоненты, позволяющие в работающей программе осуществлять выбор файлов. Delphi-диалоги выбора файла позволяют указать программе, с каким файлом мы хотим работать.

На вкладке Dialogs палитры компонентов находятся компоненты Delphi открытия файла OpenDialog и компонент сохранения файла SaveDialog (рис. 5.2).

Все компоненты, находящиеся на этой вкладке, невизуальные, т. е. при переносе на форму в окне работающей программы их не видно. Они видны только на этапе конструирования в окне формы. Компонент OpenDialog позволяет показать в программе стандартное окно

диалога открытия файла. Компонент SaveDialog выводит окно диалога сохранения файла.

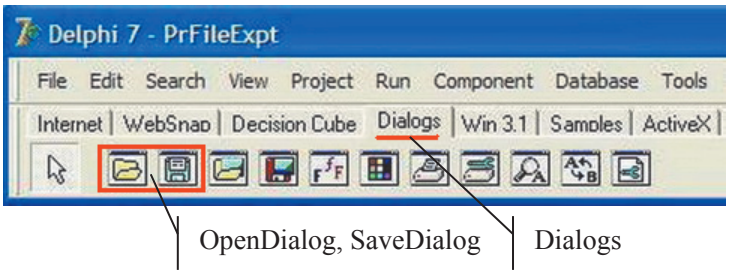


Рис. 5.2. Диалоговые компоненты

Delphi-диалоги выбора файла сами по себе ничего не делают, а только предоставляют настройки, сделанные пользователем при выборе файла. Самый важный метод Delphi-диалогов — Execute. Он срабатывает в момент нажатия кнопки «Открыть» или «Сохранить» в окне выбора файла. Для примера введем в программу возможность выбора файла для загрузки в редактор Memo1 и сохранения после редактирования. В Edit1 можно указать имя файла.

Размещаем на форме оба диалога, текстовый редактор Memo и четыре кнопки BitButton. В свойство Caption кнопок записываем: у одной — «Открыть», другой — «Сохранить», третьей — «Удалить» и четвертой — «Выход» (рис. 5.3).

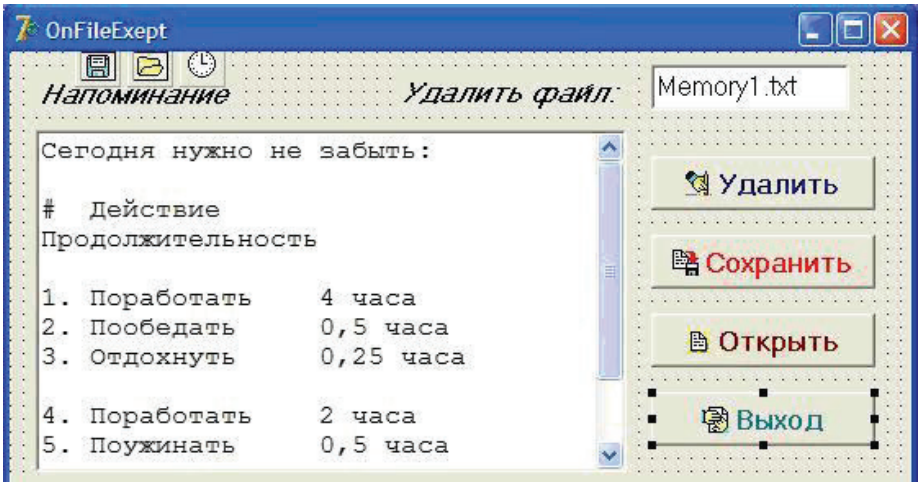


Рис. 5.3. Обработка файлов

В обработчике OnClick кнопки «Открыть» пишем:

```
if OpenDialog1.Execute then
Memo1.Lines.LoadFromFile (OpenDialog1.FileName);
```

В результате выбора файла свойство FileName компонента OpenDialog получает значение полного адреса выбранного файла, который вставляем в функцию загрузки файла компонента Memo. Этот способ хорош, когда выражение записывается в одну строку. Но нерационален, если программа использует несколько раз выражение OpenDialog1.FileName. В Delphi для такого случая есть так называемый оператор присоединения with. Он используется для любых объектов, имеющих длинный «хвост» из свойств, которые приходится записывать многократно. Вот как он записывается:

```
with Объект do
  begin
  end;
```

Свойства Объекта внутри логических скобок begin/end можно записывать непосредственно. Допускается перечислять через запятую несколько объектов. Естественно, в случае когда внутри скобок находится один оператор, они необязательны. Перепишем фрагмент загрузки файла с использованием оператора присоединения:

```
with OpenDialog1, Memo1 do
  if Execute then
    Lines.LoadFromFile (FileName);
```

Запись получается более компактной.

Так как свойства компонентов OpenDialog и SaveDialog одинаковы, сохранение текста выглядит абсолютно аналогично. Создаем обработчик нажатия кнопки «Удалить» и пишем:

```
sFile:= Edit1.Text;
AssignFile (FileName, sFile);
Erase (FileName);
```

Наконец, для кнопки «Сохранить» пишем:

```
Memo1.Lines.SaveToFile (OpenDialog1.FileName);//Сохраняем туда,
откуда считали
```

При работе этих фрагментов можно заметить, что выбирать приходится из всех файлов в нужной директории. Удобнее видеть только, например, текстовые файлы или другой тип файлов по нашему выбору. Для этого используются фильтры. Настраивается свойство Filter

наших компонентов в Инспекторе Объектов. Его выбор откроет редактор фильтров (рис. 5.4):

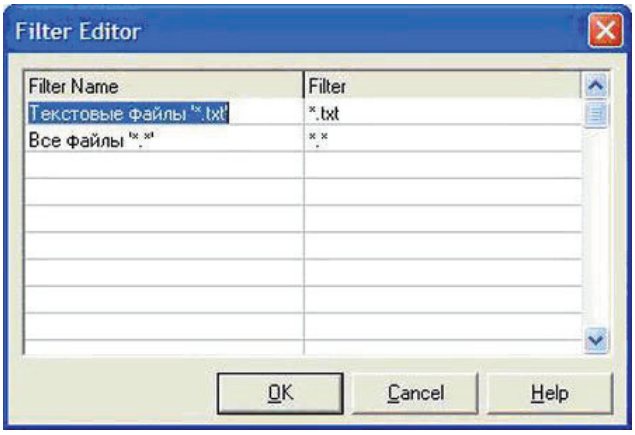


Рис. 5.4. Редактор фильтра файлов

В колонке FilterName записываем имена фильтров, в колонке Filter — список масок файлов, разделенных точкой с запятой. Маска файла в данном случае выглядит как *. расширение_файла; звездочка означает, что выбираются файлы с любыми именами, подходящие по расширению.

Свойство Delphi-диалогов Title позволяет записать в заголовок нужную фразу. Если оставить его пустым, то в заголовке будут стандартные «открыть» или «сохранить».

Свойство InitialDir позволяет в момент открытия оказаться в нужной директории. Оно доступно как на этапе «конструирования», так и при выполнении программы.

5.3. Исключения

Объектное программирование облегчает обработку ошибок времени выполнения (исключений) самим приложением. Тогда исключение не нарушает нормальный ход событий вашего кода. Фактически удаляя проверку ошибок и их обработку из основного алгоритма, исключения упрощают код.

Выделяя потенциально опасный, защищенный блок, вы определяете специальные отклики на исключения, которые могут быть в этом

блоке. Когда происходит исключение в защищенном блоке, управление немедленно передается на определенный вами отклик. Затем происходит выход из блока.

5.3.1. Обработка исключений

Синтаксис оператора обработки исключений содержит в начале новое ключевое слово try и заканчивается словом end. Слово try обозначает начало защищенного блока кода программы. Есть два типа защищенных блоков.

```
try                    //тип 1
{защищенный блок}
finally
{выполняется всегда}
end
```

Код, расположенный во второй части finally выполняется в любом случае. Такой тип оператора используется для безусловного освобождения ресурсов, занятых приложением, даже при возникновении исключения.

```
try                    //тип 2
{защищенный блок}
except
{выполняется при исключении}
end
```

Второй тип оператора используется для обработки конкретной исключительной ситуации самим приложением. Т. е. в блоке except можно:

- принять меры по исправлению ошибки пользователя;
- вывести сообщение о проблеме;
- предложить варианты решения проблемы.

В разделе except можно определить тип исключения. Тогда реакция будет соответствовать конкретному исключению. Целенаправленную реакцию на исключение обеспечивает оператор on ... do:

```
on класс_исключения do оператор
```

Оператор содержит собственно код обработки данной ошибки. Классы исключений смотрите в приложении 1.

5.3.2. Тестирование и отладка приложения

Если приложение выполняется из среды Delphi, нужно отключить отладчик, сняв флажок Integrated debugging (рис. 5.5) по команде меню Tools|Debugger Options, и реакция на исключение будет передана приложению.

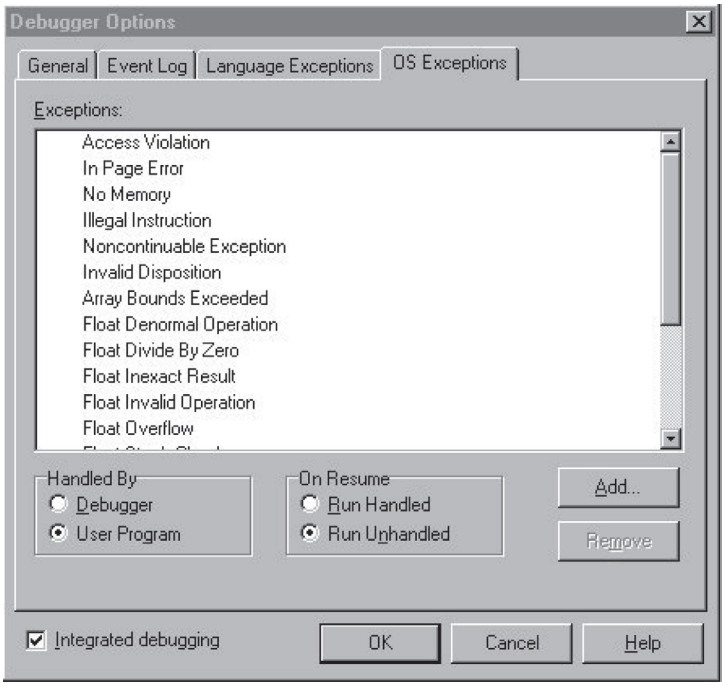


Рис. 5.5. Некоторые свойства отладчика

Второй вариант тестирования приложения — запустите исполняемый файл.exe приложения.

5.4. Исключения при обработке файлов

При обработке файлов возникают исключения класса EInOutError. Исключение EInOutError происходит при ошибках ввода-вывода, если включена директива компилятора {$I+}. Оно отвечает за ошибки ввода-вывода файлов. В поле ErrorCode экземпляра исключения содержится код ошибки ввода-вывода. Эта исключительная ситуация мо-

жет возникнуть, только если программа скомпилирована с директивой {$I} (она используется по умолчанию). Коды ошибок следующие: 2 — файл не найден, 3 — неверное имя файла, 4 — слишком много открытых файлов, 5 — доступ запрещен, 100 — конец файла, 101 — диск переполнен и 106 — некорректный ввод. Также допустимы другие коды ошибок.

Задание 1. Разместите компонеты проекта приложения на форме (рис. 5.6) и задайте их свойства.

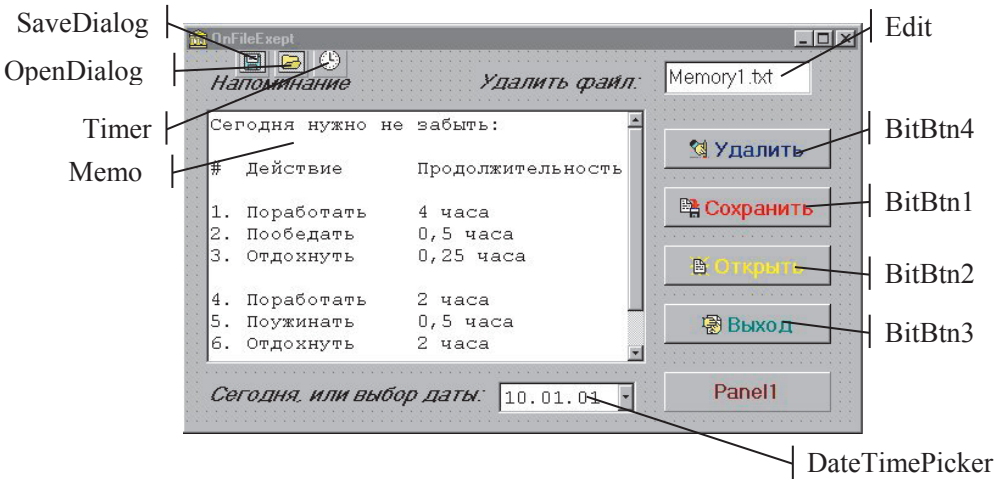


Рис. 5.6. Форма проекта обработки файла

Задание 2. Введите коды обработки событий элементов управления формы.

1. Метод обработки события OnClick кнопки BitBtn1 с надписью «Сохранить».

```
procedure TOnFileExept.BitBtn1Click (Sender: TObject);
{Событие для кнопки «Сохранить»}
begin                                {Заголовок окна диалога}
  SaveDialog1.Title:= Format ('Сохранение данных на день %s',
    [DateToStr (DateTimePicker1.Date)]);
  SaveDialog1.FileName:= '*.txt';
  if SaveDialog1.Execute then      {Выбор – сохранить файл}
  begin                            {Заполнение списка строк}
    with TstringList.Create do
      try
```

```
        Add (Format ('Напоминание дел, намеченных в день %s',
            [DateToStr (DateTimePicker1.Date)]));
        Add (#13#10); {Переход на новую строку}
        Add (Format ('%s', [Memo1.Text])); {Текст из окна Memo}
                                          {Текущая дата и время}
       Add (Format ('Справка выдана: %s', [DateTimeToStr (Now)]));
        SaveToFile (SaveDialog1.FileName); {Записать все в файл}
        Edit1.Text:= SaveDialog1.FileName;{Показать имя файла}
      finally
        Free;
      end;
    end;
end;
```

2. Метод обработки события OnClick кнопки BitBtn3 с надписью «Выход».

```
procedure TOnFileExept.BitBtn3Click (Sender: TObject);
begin                  {Событие для кнопки «Выход»}
  OnFileExept.Hide;    {Спрятать форму}
  OnFileExept.Close;   {Удалить форму из памяти}
end;
```

3. Метод обработки события OnClick кнопки BitBtn3 с надписью «Открыть».

```
procedure TOnFileExept.BitBtn2Click (Sender: TObject);
begin           {Заголовок окна диалога}
  OpenDialog1.Title:= Format ('Чтение данных в день %s',
        [DateToStr (DateTimePicker1.Date)]);
  OpenDialog1.FileName:= '*.txt';
  if OpenDialog1.Execute then
    begin
      {Считать текст из файла в окно Memo}
      Memo1.Lines.LoadFromFile (OpenDialog1.FileName);
      Edit1.Text:= OpenDialog1.FileName;{Показать имя файла}
   end;
end;
```

4. Метод обработки события OnTimer кнопки таймера Timer1.

```
procedure TOnFileExept.Timer1Timer (Sender: TObject);
begin           {Текущее время на панели}
Panel1.Caption:= TimeToStr (Time);
end;
```

5. Метод обработки события OnClick кнопки BitBtn4 с надписью «Удалить» проверяет возможные исключения при удалении файла.

```
procedure TOnFileExept.BitBtn4Click (Sender: TObject);
var MemFile: File; sFile: string [80]; s: string;
begin {Обработка исключения оператором on… do}
  sFile:= Edit1.Text;
  AssignFile (MemFile, sFile);
  try
    Erase (MemFile);
  except
    on IO: EInOutError do
      begin
        Case IO.errorcode of
          2: s:= 'Файл '''+sFile+''' не найден';
          3: s:= 'Ошибочное имя файла '''+sFile+'''';
          4: s:= 'Слишком много открытых файлов';
          5: s:= 'Файл '''+sFile+''' не доступен';
        100: s:= 'Достигнут конец файла '''+sFile+'''';
        101: s:= 'Диск переполнен при работе с файлом '''
                                          +sFile+'''';
        106: s:= 'Ошибка ввода при работе с файлом '''
                                          +sFile+'''';
      end;
     ShowMessage (s);
    end;
  end;
end;
```

Задание 3. Тестируйте и отладьте проект.

1. Выполните приложение.

Run| Run.

2. После компиляции и выполнения появится окно приложения. По очереди нажимайте кнопки и анализируйте результаты.

5.4.1. Содержание файла напоминания

```
Напоминание дел, намеченных в день 10.01.01
Сегодня нужно не забыть:
# Действие      Продолжительность
```

```
1. Поработать   4 часа
2. Пообедать    0,5 часа
3. Отдохнуть    0,25 часа
4. Поработать   2 часа
5. Поужинать    0,5 часа
6. Отдохнуть    2 часа
Справка выдана: 10.01.0117:10:16
```

Рис. 5.7 показывает результаты выполнения приложения.

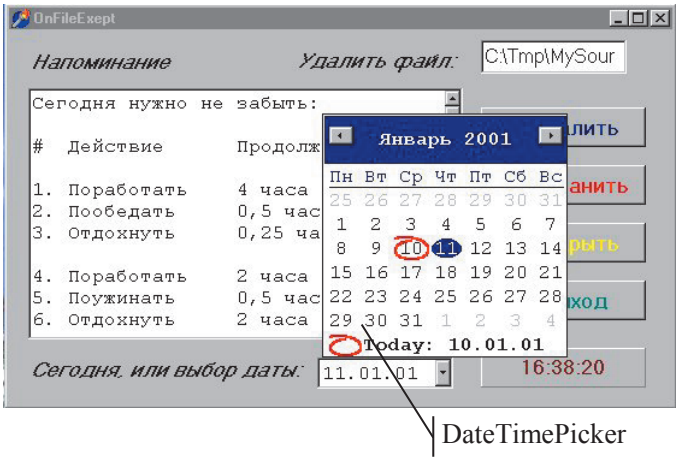


Рис. 5.7. Окно приложения обработки файла

Актуальность справки обеспечивает компонент DateTimePicker.

5.5. Приложение для работы с двумя таблицами

БД как правило состоят из нескольких таблиц. Создадим приложение для работы с двумя таблицами базы данных. Необходимо установить связи между таблицами так, чтобы одна была главной, а вторая — подчиненной.

5.5.1. Свяжем два набора данных

В одной форме можно связать два набора данных (главный и подчиненный) так, чтобы в подчиненном наборе показывались записи, соответствующие текущей записи в главном наборе.

1. Поместите на форму два компонента: Table (с именем Table1) и DataSource (имя dsTovar) для работы с таблицей товаров. С помощью свойств DatabaseName и TableName настройте компонент Table1 на работу с таблицей Tovar.DB и установите в его свойстве Active значение True, а с помощью свойства DataSet компонента dsTovar свяжите его с компонентом Table1.

2. Разместите на форме компонент DBGrid (имя DBGrid1) и установите в его свойство DataSource значение dsTovar. Чтобы пояснить, содержимое каких таблиц отображают компоненты DBGrid, поместите над ним компонент Label и задайте в его свойстве Caption значение «Товары:».

3. Добавьте на форму еще одну пару компонентов Table (с именем Table2) и DataSource (имя dsPrihod) для работы с таблицей прихода товаров. С помощью свойств DatabaseName и TableName настройте компонент Table2 на работу с таблицей Prihod.DB и установите в его свойство Active значение True, а с помощью свойства DataSet компонента dsPrihod свяжите его с компонентом Table2.

4. Вставьте в форму компонент DBGrid (имя DBGrid2) и установите в его свойстве DataSource значение dsPrihod. Для того чтобы пояснить, содержимое каких таблиц отображают компоненты DBGrid, поместите над ним компонент Label2 и установите в его свойстве Caption значение «Приход товаров:» (рис. 5.8). Переименуйте название формы на имя «Склад».

Если сейчас запустить программу на исполнение, обе таблицы будут независимы друг от друга.

5. Индексируйте связывающее поле tovar в таблицах (TovarIndT, TovarIndP). Чтобы содержимое подчиненного набора соответствовало выбору записи в главном наборе, подчиненную таблицу нужно связать с главной. Для этого раскройте список выбора в свойстве MasterSource таблицы Table2 и выберите единственное имеющееся в нем значение dsTovar. Затем щелкните по правой части строки MasterFields в окне инспектора объектов и по появившейся в ней кнопке с тремя точками, чтобы раскрыть окно редактора связей.

6. Раскройте список Available Indexes в верхней части этого окна и выберите индекс TovarIndP — в окошке Detail Fields появится имя поля связи Tovar. Выберите это же поле в окошке MasterFields и нажмите кнопку Add — связь установлена (рис. 5.9) Tovar –> Tovar.

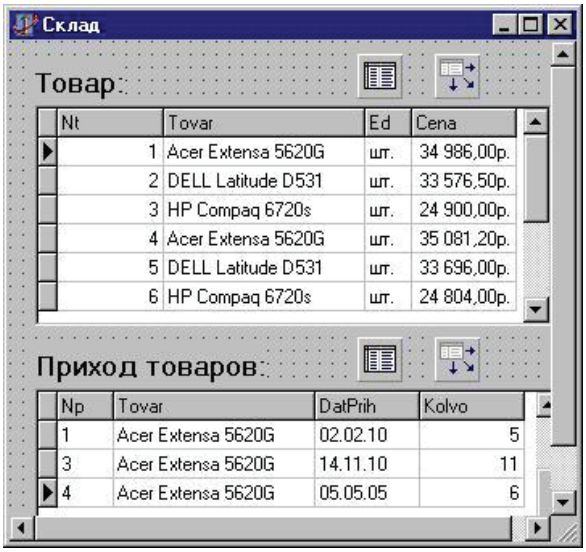


Рис. 5.8. Форма с главным и подчиненным наборами данных во время проектирования

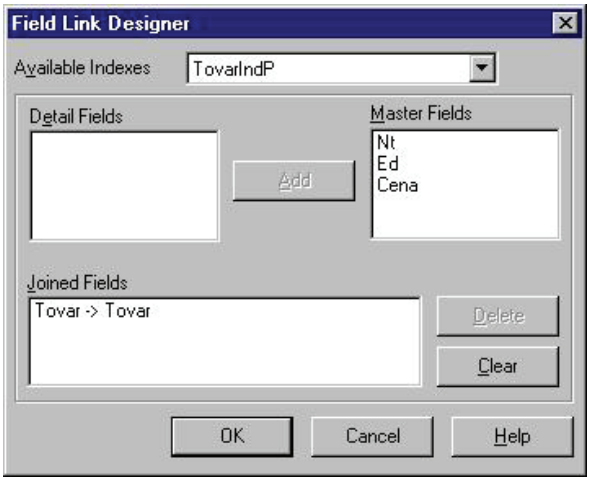


Рис. 5.9. Окно редактора связей

7. Закройте окно редактора связей кнопкой OK и запустите приложение. Ввод и изменение данных в таблице Prihod можно производить с помощью компонента DBGrid2 (рис. 5.10). При этом перемещение указателя в главной таблице приводит к автоматической смене информации в отображаемых данных подчиненной таблицы. Заполните таблицу Prihod, вводя «Даты прихода» товара и его количество.

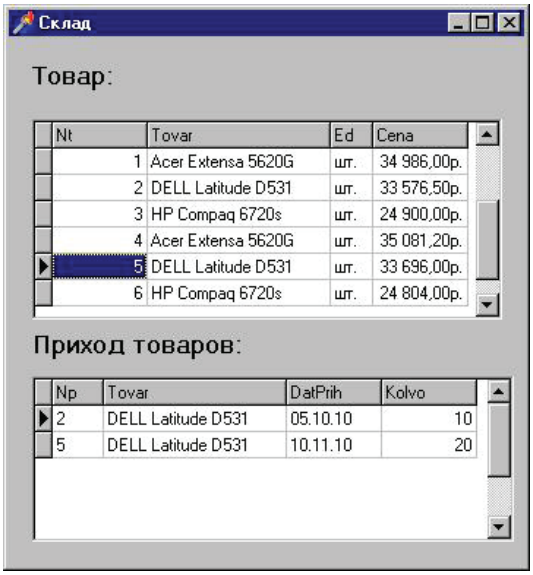


Рис. 5.10. Вид приложения, содержащего таблицы с главным и подчиненным наборами

5.5.2. Правильный выбор файлов

1. Добавьте в форму проекта: диалог открытия файла, поле ввода, две кнопки (рис. 5.11). Закройте таблицы в проекте. Откройте первую и вторую таблицы в диалоге при выполнении приложения.

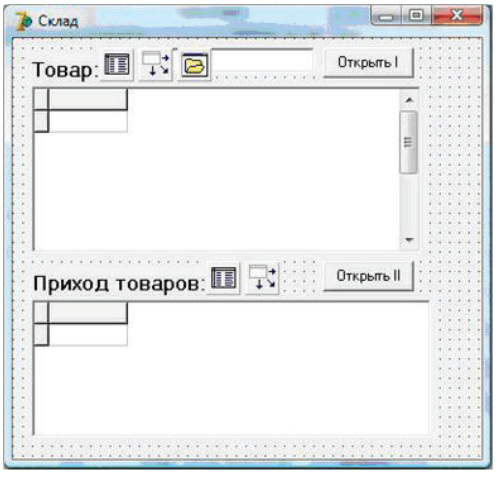


Рис. 5.11. Вид приложения, содержащего таблицы с главным и подчиненным наборами

2. Обработайте возможные исключения при открытии таблиц. Например, для первой таблицы.

```
procedure TForm1.Button1Click (Sender: TObject);
var fFileName: string;
begin
  OpenDialog1.FileName:= '*.db';
  try
    if OpenDialog1.Execute then With Table1 do
    begin
      Active:= False; {Закрыть таблицу БД}
      fFileName:= OpenDialog1.FileName;
      Edit1.Text:=fFileName;{Показать имя файла}
      DatabaseName:= ExtractFilePath (fFileName);
      TableName:= ExtractFileName (fFileName);
      Active:= True {Открыть таблицу БД}
    end
   except
    ShowMessage ('Ошибка открытия I таблицы')
  end
end;
```

3. Аналогичный код введите для того, чтобы открыть вторую таблицу кнопкой Button2.

4. Тестируйте и отладьте проект.

5.6. Файловые подпрограммы Delphi

В системе помощи Delphi существует понятие «подпрограммы управления файлами». Процедуры и функции, входящие в эту категорию, находятся в модулях System и SysUtils (каталог Source\Rtl\Sys).

Местонахождение процедур и функций определяется следующим образом. Если в подпрограмме используется файловая переменная, то она входит в модуль System. Если используется дескриптор или имя файла в виде строки, то подпрограмма описана в модуле SysUtils. Смотрите приложение 3.

5.6.1. Применение файловых подпрограмм

Задание 1. Следующий пример использует кнопку Button1 с надписью «Сохранить», сетку строк StringGrid и диалог сохранения файла SaveDialog на форме (рис. 5.12). Когда нажимается кнопка, у пользователя запрашивается имя файла. Когда пользователь нажимает OK, содержание StringGrid записывается в определенный файл. Дополнительная информация записывается в файл так, чтобы она могла легко читаться функцией FileRead.

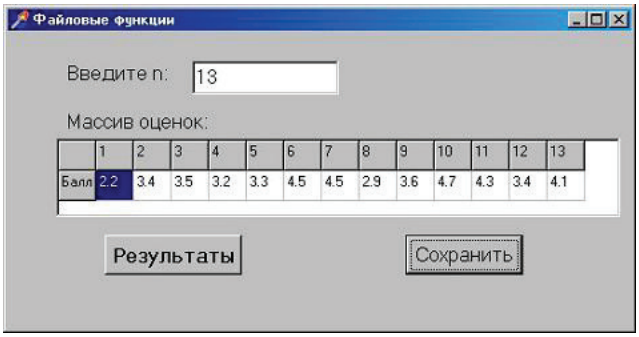


Рис. 5.12. Сохранение файла

Задание 2. Определите метод обработки события OnClick кнопки Button1 с надписью «Сохранить».

```
procedure TForm1.Button1Click (Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X: Integer;
  Y: Integer;
begin
  if SaveDialog1.Execute then
  begin
   if FileExists (SaveDialog1.FileName) then begin
    BackupName:= ExtractFileName (SaveDialog1.FileName);
    BackupName:= ChangeFileExt (BackupName, '.BAK');
    if not RenameFile (SaveDialog1.FileName, BackupName) then
     raise Exception.Create ('Нельзя создать файл backup.');
```

```
        end;
        FileHandle:= FileCreate (SaveDialog1.FileName);
        {Впишет в файл число строк и столбцов в сетке}
        FileWrite (FileHandle,
          StringGrid1.ColCount, SizeOf (StringGrid1.ColCount));
        FileWrite (FileHandle,
          StringGrid1.RowCount, SizeOf (StringGrid1.RowCount));
        for X:= 0 to StringGrid1.ColCount – 1 do
        begin
         for Y:= 0 to StringGrid1.RowCount – 1 do begin
             {Впишет длину каждой строки, сопровождаемую строкой
непосредственно}
           StringLen:= Length (StringGrid1.Cells [X, Y]);
           FileWrite (FileHandle, StringLen, SizeOf (StringLen));
           BackupName:= StringGrid1.Cells [X, Y];
           FileWrite (FileHandle, BackupName,
                Length (StringGrid1.Cells [X, Y]));
         end;
       end;
       FileClose (FileHandle);
     end;
   end;
```

Задание 3. Рассмотрите пример с генерацией одномерного массива его сохранением в файл и чтением полученного файла функцией FileRead.

Глава 6.
Управление базами данных

Реализуем систему управления базой данных в форме многооконного приложения. Приложение будет содержать элементарные операции с конкретной таблицей. Будут реализованы функции создания, редактирования, просмотра таблицы и организован редактор запросов на языке SQL. Внедрено четыре связанных формы. Реальное приложение должно содержать дополнительные команды меню, окна и т. д.

6.1. Шаг 1. Начало и конец работы приложения в меню

1. File|New Application
2. Сохраните все файлы приложения: File|Save All
Отправьте файлы в свою папку и укажите их имена.
3. В Инспекторе Объектов (ИО) задайте свойства формы
Caption = Таблица цен (введите заголовок по названию своей темы)
Hint = Быстрее
ShowHint = true
4. Палитра компонентов (ПК), страница Standard. Выберите компонент MainMenu разместите в левом нижнем углу формы. В ИО задайте свойства компонента.
Items = Щелкните кнопку с многоточием [...]
Появится окно Конструктора Меню (КМ). Встаньте на темный прямоугольник первого пункта меню и задайте в ИО его свойства:
Caption = &Файл (Буква Ф будет подчеркнута и доступна как Alt+Ф)
Name = File1
Добавим пункт выпадающего меню «Выход». В КМ щелкните по пункту «Файл», ниже появится прямоугольник подменю. Щелкните в прямоугольнике, он потемнеет. В ИО задайте свойства подменю:

Caption = В&ыход
Name = Exit1

Добавим раздел меню «Справка» с подпунктом «О программе». Выберите прямоугольник справа от пункта меню «Файл». В ИО задайте свойства:

Caption = Спр&авка
Name = Help1

Ниже проведем горизонтальную разделительную черту.

Caption = — (один символ тире!)
Name = Help2

Ниже черты добавим подраздел меню «О программе»

Caption = &О программе
Name = About1

Закройте КМ кнопкой с крестиком ☒.

В верхней строке формы появилось меню.

Введем код выхода из программы при выборе соответствующей команды меню. Выберите в меню на форме пункт «Выход», откроется окно редактора кода, курсор стоит на месте ввода. Добавьте нужный оператор Close.

```
procedure TForm1.Exit1Click (Sender: TObject);
begin
  Close;                    {Закрыть окно}
  Application.Terminate;    {Завершить программу}
end;
```

8. Введем код справки о программе, ее авторе, дате создания. Выберите в меню на форме подпункт «О программе», откроется окно редактора кода, курсор стоит на месте ввода. Добавьте нужный код.

```
procedure TForm1.About1Click (Sender: TObject);
begin
  ShowMessage ('Проект приложения по теме… создан… (Дата).'+
  ' Выполнил студ… Руководитель доц. каф. ВТ Неудачин И.Г. ');
end;
```

9. Добавьте на форму компонент выпадающего контекстного меню PopupMenu1. Свяжите его с формой, задав в ИО свойство формы

PopupMenu = PopupMenu1

10. Определим команды выпадающего меню. Для этого выберите компонент PopupMenu1, перейдите в ИО, найдите свойство Items. Задайте два пункта меню со свойствами:

Caption = В&ыход
Name = Exit
Caption = &О программе
Name = About

11. Свяжите событие OnClik первого пункта выпадающего меню Exit с уже существующим обработчиком события Exit1Click на странице событий ИО. Свяжите событие OnClik второго пункта меню About с обработчиком события About1Click на странице событий ИО.

12. Сохраните все файлы проекта приложения: File|Save All.

13. Компилируйте и выполните программу: Run|Run.

14. Исправьте обнаруженные компилятором ошибки.

15. Тестируйте программу. Вызовите пункты меню и убедитесь в их работоспособности:

Справка| О программе (испытайте также комбинацию клавиш Alt+О).

Файл| Выход (испытайте также комбинацию клавиш Alt+ы).

Нажмите правую кнопку мыши в окне приложения и проверьте команды выпадающего меню.

В окне приложения указатель мыши должен вызывать всплывающую подсказку «Быстрее». Проверьте заголовок окна по названию своей темы.

16. В случае возникновения ошибки при выполнении программы выгружайте ее из памяти командой Run|Program Reset из меню Delphi.

6.2. Шаг 2. Создание таблицы базы данных

1. Создадим новый пункт выпадающего меню: Файл|Создать.

Вызовем Конструктор Меню (КМ) другим способом. Выберите на форме компонент MainMenu и щелкните по нему два раза левой кнопкой мыши (л. к. м.). В открывшемся окне КМ выберите пункт «Выход», щелкните на нем правой кнопкой (п. к. м.) мыши для вызова команды локального меню Insert. В Инспекторе Объектов (ИО) задайте свойства:

Caption = &Создать
Name = CreateTab1
Закройте КМ кнопкой ☒.

2. Разместим в левом нижнем углу формы (рис. 6.1) компонент Table страницы BDE Палитры Компонентов.

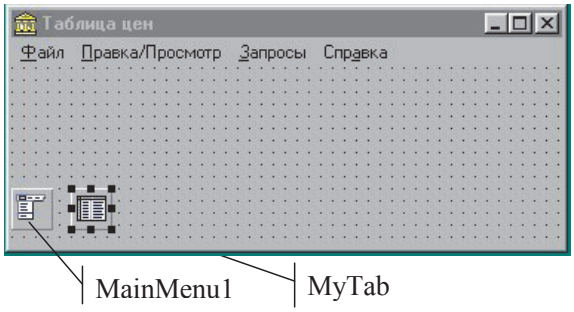


Рис. 6.1. Форма, содержащая компонеты MainMenu и Table

Свойства компонента:

Name = MyTab

DatabaseName =.\(текущий каталог)

TableName = Tablxx (имя файла с таблицей, лучше свое имя)

3. Опишем структуру таблицы в Редакторе Полей (РП).

Выберите на форме компонент Table и сделайте два щелчка л. к. м.

3.1. Откроется окно Fields Editor (РП), сделайте щелчок п. к. м., появится локальное меню New Field... В рамке Field properties задайте свойства первого поля таблицы:

Name = имя поля 1 (латинские буквы и цифры)

Type = выберите тип данных поля из списка

Если тип String, укажите свойство

Size = размер поля в символах

Нажмите кнопку OK. Откроется окно РП. Выберите создаваемое поле 1. В ИО задайте свойство поля 1:

KeyFields = имя этого поля, если оно ключевое.

3.2. Повторяйте пункт сценария 3.1 для каждого поля таблицы.

3.3. Закройте окно РП кнопкой ☒.

4. Введите код, выполняемый по команде меню Файл| Создать.

Щелкните л. к. м. по этому пункту на форме, откроется окно редактора кода, курсор стоит на месте ввода| Добавьте нужный оператор:

```
procedure TForm1.CreateTab1Click (Sender: TObject);
begin
  if MessageDlg ('Если файл был раньше создан, ',
```

```
      ' сейчас он уничтожится. Создать файл?',
      mtConfirmation'mbYes, mbNo], 0) = mrYes
    then MyTab.CreateTable; {Создать таблицу}
end;
```

5. Сохраните все файлы проекта приложения: File|Save All

6. Компилируйте и выполните программу. Исправьте обнаруженные компилятором ошибки. Run|Run

7. Тестируйте программу. Вызовите пункт меню Файл|Создать и убедитесь в его работоспособности. Затем выйдите из программы: Файл|Выход. Проверьте наличие на диске в соответствующей папке своего файла tablxx.DB. Если был указан другой псевдоним или каталог, ищите его там.

6.3. Шаг 3. Просмотр, заполнение, редактирование таблицы

1. Вставим новый пункт главного меню «Правка/Просмотр».

Выберите на форме компонент MainMenu, щелкните по нему два раза л. к. м. Откроется окно КМ. Выберите пункт «Справка», щелкните на нем п. к. м. для вызова локального меню| Insert| В Инспекторе Объектов (ИО) задайте свойства:

Caption = &Правка/Просмотр

Name = EditTab1

Закройте КМ кнопкой ☒.

2. Настроим компонент Form1.MyTab для работы. Зададим выводимые заголовки полей (столбцов) таблицы в Редакторе Полей (РП). Сделайте два щелчка л. к. м. по компоненту Table на форме. Войдите в появившееся окно РП. Откройте щелчком п. к. м. локальное меню, выполните команду Add fields... и OK. Выбирайте по очереди поля и в ИО задавайте их свойство:

DisplayLabel = название соответствующего поля по-русски

3. Поместите на форму компонент DataSource со страницы Data Access и задайте его свойство DataSet = MyTab

4. Свяжем с этим разделом меню новое окно. Новое окно программы создаст новая форма Form2 проекта приложения. Добавим в проект новую форму.

File|New Form| В ИО задайте свойство Caption = Правка/Просмотр

5. Поместим на форму Form2 компонент DBGrid со стр. DataControls. Задайте его свойство DataSource = Form1.DataSource1 (рис. 6.2).

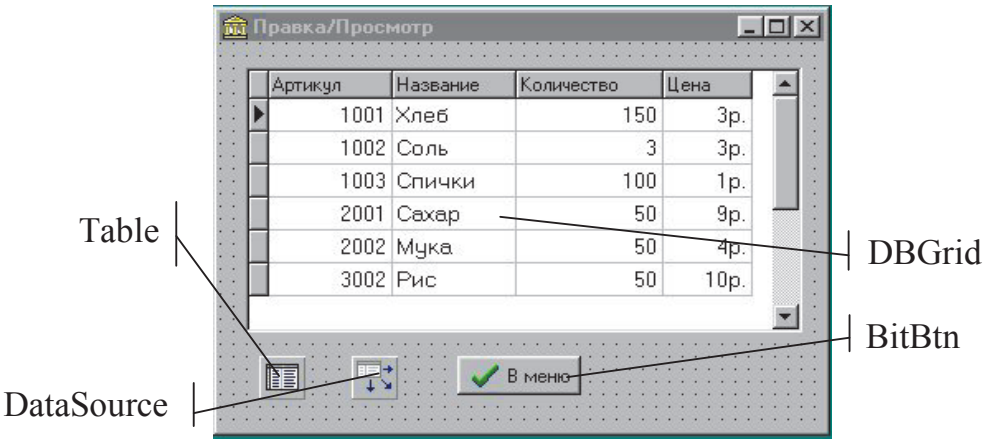


Рис. 6.2. Форма Form2 для редактирования и просмотра таблицы

После этого таблица должна появиться в компоненте DBGrid на форме, если она создана, находится в рабочем каталоге и открыта (свойство Active = True, рис. 6.2).

6. Установим связь двух форм (будущих окон программы). Опустите на форму кнопку BitBtn со стр. Additional ПК. Кнопка содержит картинку и можно выбирать цвет ее шрифта. В ИО задайте свойства:

Kind = bkOK

Caption = В меню

Font = вид, размер, начертание шрифта

Щелкните л. к. м. по этой кнопке на форме два раза, откроется окно редактора кода, курсор стоит на месте ввода. Добавьте нужные операторы:

```
procedure TForm2.BitBtn1Click (Sender: TObject);
begin
  Form1.MyTab.Close;   {Закроем таблицу, чтобы ее создавать}
  Form2.Hide;          {Спрятать форму}
  Form2.Close;         {Удалить форму из памяти}
  Form1.Show;          {Показать форму}
end;
```

7. Если окно Form2 закроется кнопкой ☒, предусмотрим обработку события OnClose, для того чтобы не потерять доступ к приложению:

```
procedure TForm2.FormClose (Sender: TObject; var Action:
          TCloseAction);
begin
  Form1.MyTab.Close; {Закроем таблицу, чтобы ее создавать}
  Form1.Show; {Показать форму}
end;
```

8. Вернемся в первую форму через главное меню Delphi — команда View|Forms... и установим способ вызова созданной формы через пункт меню «Правка/Просмотр».

Щелкните л. к. м. по этому пункту на форме, откроется окно редактора кода, курсор стоит на месте ввода. Добавьте нужные операторы:

```
procedure TForm1.EditTab1Click (Sender: TObject);
begin
  Form1.MyTab.Open;    {Откроем таблицу для обработки}
  Form1.Hide;          {Спрятать форму}
  Form2.Show;          {Показать форму}
end;
```

9. Сохраните все файлы приложения: File|Save All

10. Компилируйте и выполните программу. Исправьте обнаруженные компилятором ошибки. Run|Run. На появившееся предложение подключать модули отвечайте Yes.

11. Тестируйте программу. Вызовите пункт меню «Правка/Просмотр» и убедитесь в его выполнении. Заполните таблицу данными (не меньше 10 записей). Последняя запись сохранится при переходе в следующую строку таблицы. Затем выйдите из окна и раздела меню с помощью кнопки «В меню». Закончите программу через меню Файл|Выход.

6.4. Шаг 4. Запросы данных из таблицы

Организуем запросы к таблице на языке SQL в вашем проекте.

Оператор select

Краткие сведения об операторе select структурированного языка запросов SQL для получения информации из таблицы.

Формат оператора:

```
select список_выражений from файл_таблицы
```

```
where условие
order by Поле
```

Список выражений задает выводимую из таблицы информацию, в т. ч. с вычислениями и функциями полей sum, count, average, max, min. Файл таблицы содержит исходную информацию. Условие определяет критерий отбора записей из файла таблицы. Поле в формате задает сортировку выводимых записей в соответствии с возрастанием значений этого поля. Не все части оператора обязательны. Пример: таблица Tablxx содержит поля с именами a, b, c, d.

```
SELECT * FROM Tablxx {Вывести все поля – это комментарий}
SELECT b, d FROM Tablxx      {Вывести поля b, d}
SELECT count (*) FROM Tablxx {Количество записей}
SELECT Max (c) FROM Tablxx   {Максимальное значение c}
SELECT 'Сумма цен = ', Sum (d) FROM Tablxx {Сумма значений поля d}
SELECT 'Цена ', b,' = ', c*d FROM Tablxx {Произведения c*d}
SELECT b, d FROM Tablxx {Выборка внутри диапазона 30<d<60}
WHERE 30<d AND d<60 ORDER BY d
```

Конечно, вы помните имя своей таблицы и имена ее полей!

1. Вставим новую форму Form3 в проект File|New Form. В ИО задайте свойство Form3.

Caption = Запросы (это заголовок формы и окна)

2. Опустите на форму три кнопки BitBtn со страницы Additional ПК. Первую кнопку поместите внизу слева, определите ее свойства:

Kind = bkOK (из списка), Caption = &Выполнить

Вторую кнопку вставьте внизу справа, определите ее свойства:

Kind = bkCancel (из списка), Caption = &Отмена

Третью кнопку поместите внизу в центре, определите ее свойства:

Kind = bkOK (из списка), Caption = В &меню

3. Запросы будем вводить в окне редактора заметок Memo. Возьмите этот компонент со страницы Standard ПК. Займите им верхнюю половину формы.

Сверху сделайте надпись с помощью метки Label и ее свойства

Caption = Введите оператор запроса SQL: Lines = кнопка [...]. Введите в окно текст запроса по умолчанию:

SELECT * FROM Tablxx

4. Возьмем со страницы BDE на форму компонент Query и зададим его свойство DatabaseName =.\(рабочий каталог).

Разместим здесь же компонент DataSource1 и зададим его свойство DataSet = Query1

5. Вставим компонент DBGrid со стр. Data Controls в форму. Зададим его свойство DataSource = DataSource1. Займите им нижнюю половину формы.

Сверху сделайте надпись с помощью метки Label и ее свойства Caption = Результаты запроса:

6. Коды обработки событий вводятся в окне редактора, который вызывается двойным щелчком л. к. м. по соответствующей кнопке.

Кнопка «Выполнить» введет запрос на обработку:

```
procedure TForm3.BitBtn1Click (Sender: TObject);
begin
  Query1.Close;                  {Закрыть компонент}
  Query1.SQL. Clear;             {Удалить старый запрос}
  Query1.SQL. Add (Memo1.Text); {Ввести новый запрос}
  Query1.Prepare;                {Подготовить компонент}
  Query1.Open;                   {Переоткрыть компонент}
end;
```

Кнопка «Отмена» отменит набранный ранее в окне Memo запрос и удалит текст старого запроса:

```
procedure TForm3.BitBtn2Click (Sender: TObject);
begin
  Query1.Close;
  Query1.SQL. Clear;
  Memo1.Text:= '';    {Удалить текст старого запроса}
end;
```

Кнопка «В меню» вызовет главное меню вашей программы:

```
procedure TForm3.BitBtn3Click (Sender: TObject);
begin
  Form3.Hide; Form3.Close; Form1.Show;
end;
```

Если окно Form3 закроется кнопкой с крестиком ☒, предусмотрим обработку события OnClose, для того чтобы не потерять доступ к приложению:

```
procedure TForm3.FormClose (Sender: TObject; var Action: TClose-
Action);
begin Form1.Show; end;
```

7. Добавим на Form1 раздел меню «Запросы» и напишем код для вызова вновь созданной формы Form3 (рис. 6.3).

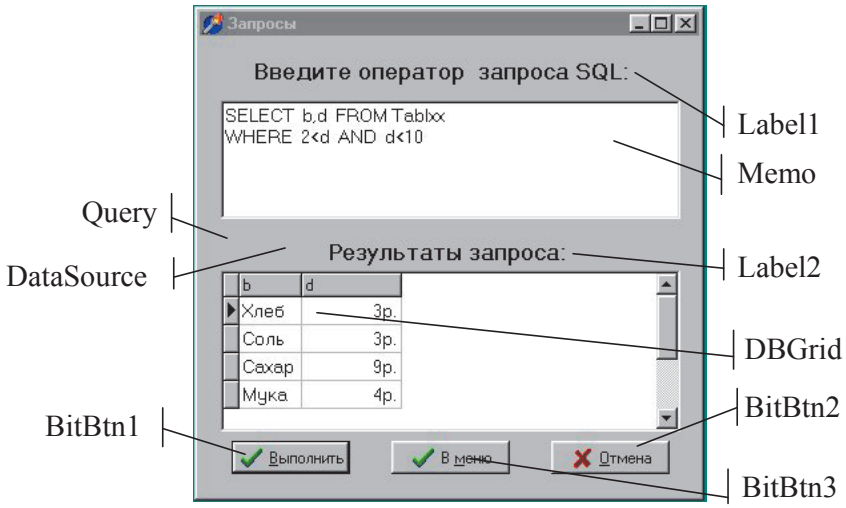


Рис. 6.3. Окно программы по форме Form3 с примером запроса

Вернемся в первую форму через главное меню Delphi — команда View|Forms... Выберите на форме компонент MainMenu, щелкните по нему два раза л. к. м. Откроется окно КМ. Выберите пункт «Справка», щелкните на нем п. к. м. для вызова локального меню Insert. В Инспекторе Объектов (ИО) задайте свойства:

Caption = &Запросы Name = QueryTab1

Закройте КМ кнопкой с крестиком ☒.

Сделайте щелчок л. к. м. на разделе «Запросы» формы. Введите нужные операторы:

```
procedure TForm1.QueryTab1Click (Sender: TObject);
begin
  Form1.Hide;   {Спрятать форму}
  Form3.Show;   {Показать форму}
end;
```

8. Сохранение, компиляцию, тестирование программы делайте по шагам 1–3.

6.5. Шаг 5. Дополнения на форме Form1 (главное окно)

Оформим некоторые дополнительные сервисы главной формы приложения, для того чтобы она была информативна. Одновременно познакомимся с новыми страницами Палитры Компонентов.

1. Установим календарь текущего месяца с выделением текущей даты.

Компонент Calendar со страницы Samples опустите на форму Form1. Задайте положение, размеры и свойства компонента. В ИО установите размер и цвет шрифта, а также всплывающую подсказку Hint = Дата.

2. Установим панель, показывающую текущее время.

Под календарь поместите компонент Panel со страницы Standard. В ИО установите размер и цвет шрифта, а также всплывающую подсказку Hint = Время.

Справа от Panel установим элемент Timer со страницы System. Осталось ввести код, показывающий текущее время компонента Timer в панели Panel. Сделайте два щелчка л. к. м. по элементу Timer. Вы попали в окно редактора для установки недостающего оператора в событии OnTimer.

```
procedure TForm1.Timer1Timer (Sender: TObject);
begin
  Panel1.Caption:= TimeToStr (Time);
end;
```

3. Познакомимся с диалоговым элементом открытия файла OpenDialog. Возьмите этот компонент со страницы Dialogs и поместите справа от таймера.

4. Добавим элементы управления диалогом внизу формы в последовательности (рис. 6.4) слева направо: Label, Edit, Button. Они находятся на странице Standard.

В ИО укажем их свойства.

Компонент Label, свойство Caption = Выбран файл:

Компонент Edit, свойство Text очистите.

Компонент Button, свойство Caption = Выбрать

Характеристики шрифтов для Label, Edit задайте по своему усмотрению.

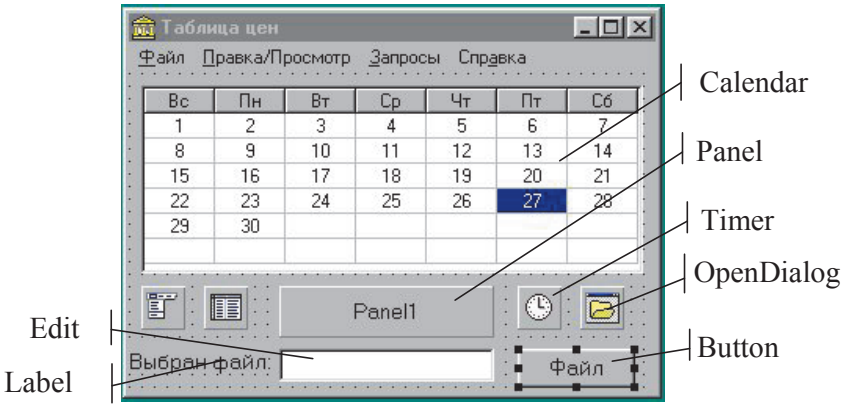


Рис. 6.4. Форма Form1 на шаге 5 с дополнительными сервисами

Сделайте два щелчка л. к. м. по элементу Button и введите код обработки события OnClick.

```
procedure TForm1.Button1Click (Sender: TObject);
begin
  OpenDialog1.FileName:= '*.DB'; {Маска поиска по умолчанию}
  if OpenDialog1.Execute then
    begin
      Edit1.Text:= OpenDialog1.FileName; {Вывод имени файла}
      MyTab.TableName:= OpenDialog1.FileName; {Выбрать БД}
    end
end;
```

5. Сохранение, компиляцию, тестирование программы вы помните по шагам проектирования 1–3.

6.6. Шаг 6. Справка о записи в таблице

Фактически на этом шаге создается простой текстовый редактор, встроенный в приложение.

6.6.1. Конструирование формы

1. Вставим новый подпункт главного меню «Файл справки» между разделами «Справка» и «О программе».

Выберите на форме Form1 компонент MainMenu1, щелкните по нему два раза л. к. м. Откроется Окно Конструктора Меню (КМ)| Выберите разделительную черту над пунктом «О программе», щелкните на нем п. к.м. для вызова команды Insert из локального меню. Встаньте на появившийся темный прямоугольник| В Инспекторе Объектов (ИО) задайте свойства:

Caption = Фа&йл справки

Name = N1

Закройте КМ кнопкой с крестиком ☒.

На форме выберите подменю «Файл справки» и щелкните по нему л. к. м. Введите код обработчика события выбор меню «Файл справки».

```
procedure TForm1.N1Click (Sender: TObject);
begin
  Form1.MyTab.ReadOnly:= True; {Запретить редактирование}
  Form1.MyTab.Open; {Откроем таблицу для оформления справок}
  Form4.ShowModal; {Модальное окно. Выход только после закрытия}
end;
```

Главное окно Form1 остается открытым.

2. Свяжем с этим разделом меню новое окно. Окно программы создаст форма Form4 проекта приложения. Добавим в проект новую форму.

File|New Form| В ИО задайте свойство Caption = Файл справки.

3. Если окно Form4 закроется кнопкой ☒, предусмотрим обработку события OnClose, для того чтобы закрыть таблицу:

```
procedure TForm3.FormClose (Sender: TObject; var Action: TClose-
Action);
begin
  Form1.MyTab.Close; {Закроем таблицу, чтобы ее создавать}
  Form1.MyTab.ReadOnly:= False; {Разрешить редактирование}
  MemHelp.Clear; {Очистим текстовое окно}
end;
```

4. Разместим сверху на форме Form4 компоненты со страницы Standard.

4.1. Слева вставьте метку Label1, со свойством Caption = Справка:

4.2. Правее вставьте (рис. 6.5) метку Label2, у которой свойство Caption = Справка о:

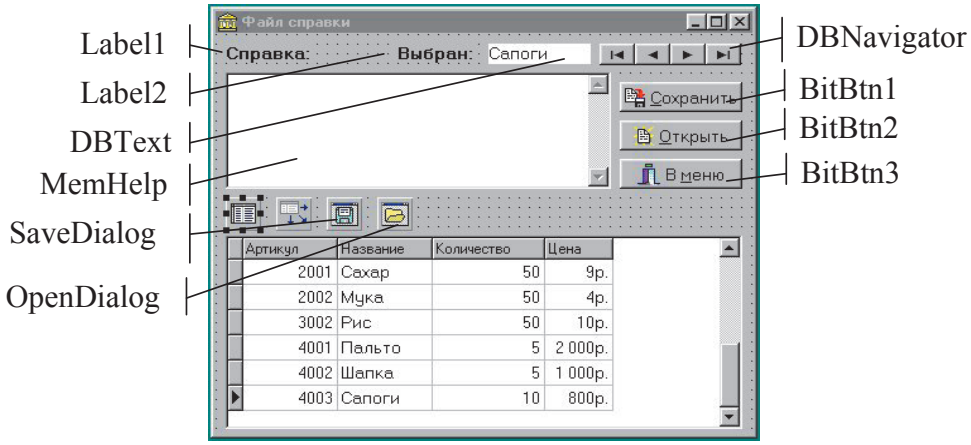


Рис. 6.5. Форма Form4 приложения обработки таблицы

4.3. Ниже поместите компонент окна справок Memo на 4/5 ширины формы (страница Standard) со свойствами:

Lines = кнопкой [...] очистите окно, закройте окно кнопкой [X].

Name = MemHelp

ScrollBars = ssVertical (выберите из списка)

Hint = Дополнения

ShowHint = True (выберите из списка)

Выберите свойства шрифтов для установленных компонентов.

5. Добавим на форму компоненты ПК со страницы Data Controls.

5.1. Справа от метки Label2 поместите окно DBText со свойствами:

DataSource = Form1.DataSource1

DataField = выберите из списка подходящее имя поля для вывода справки о выбранной строке таблицы.

Color = цвет фона (из списка)

5.2. Правее поместите элемент управления таблицей DBNavigator.

По умолчанию DBNavigator содержит большее количество средств управления, чем потребуется для этого малого приложения. Теперь вы нуждаетесь только в First, Prior, Next и Last (Первый, Предшествующий, Затем и Последний) кнопках навигатора. Свойство VisibleButtons объекта DBNavigator контролирует вывод на экран заданных кнопок навигатора. Переключите значение True на False у кнопок: nbInsert, nbDelete, nbEdit, nbPost, nbCancel, nbRefresh. Конечно, нужно привязать навигатор к источнику данных через свойство DataSource = Form1.DataSource1

5.3. Внизу формы вставьте компонент DBGrid и задайте его свойство DataSource = Form1.DataSource1

6. Со страницы Dialogs опустите на форму компоненты SaveDialog и OpenDialog.

Определим в ИО свойство Filter, нажав на кнопку [...] справа от названия свойства. Откроется окно Filter Editor. Задайте маски поиска:

FilterName	Filter
Текстовые файлы '*.txt'	*.txt
Все файлы '*.*'	*.*

Нажмите кнопку OK.

7. Правее объекта MemHelp установите три кнопки BitBtn со страницы Additional. Опишем их свойства и зададим коды событий OnClick.

Кнопка BitBtn1:

Caption = &Сохранить

Glyph = нажмите кнопку [...] справа.

Откроется окно диалога Load... Задайте путь к файлу с рисунком на кнопке: C:\Program Files\Common Files\Borland Shared\Images\Buttons\Filesave.bmp |Открыть|OK

Кнопка BitBtn2:

Caption = &Открыть

Glyph = ...\Fileopen.bmp

Кнопка BitBtn3:

Kind = bkOK (из списка)

Caption = В &меню

По очереди делайте два щелчка л. к. м. на каждой кнопке и вводите коды обработки событий OnClick.

6.6.2. Обработчики событий для трех кнопок

Определите метод обработки события OnClick для кнопки BitBtn1:

```
procedure TForm4.BitBtn1Click (Sender: TObject);
var i: integer;
begin {Заголовок окна диалога}
  SaveDialog1.Title:= Format ('Сохранение данных для%s',
          [DBText1.Field.AsString]);
  if SaveDialog1.Execute then {Выбор – сохранить файл}
  begin {Заполнение списка строк}
    with TstringList.Create do
```

```
      try
       Add (Format ('Документальная справка про%s',
              [DBText1.Field.AsString])); Add (#13#10);
       Add (Format ('%s:%s', [DBGrid1.Columns [0].Title.Caption,
              DBGrid1.Fields [0].AsString]));
       for i:= 2 to 3 do {Заголовок}
       Add (Format ('%s:%s', [DBGrid1.Columns [i].Title.Caption,
              DBGrid1.Fields [i].AsString]));{Данные}
       Add (Format ('Дополнения:%s', [MemHelp.Text]));
              {Текст Memo}
       Add (Format ('Справка выдана:%s', [DateTimeToStr (Now)]));
       SaveToFile (SaveDialog1.FileName); {Записать все в файл}
      finally
     Free;
    end;
  end;
end;
```

Определите метод обработки события OnClick для кнопки BitBtn2:

```
procedure TForm4.BitBtn2Click (Sender: TObject);
begin
  if OpenDialog1.Execute then {Считать из файла в окно Memo}
    MemHelp.Lines.LoadFromFile (OpenDialog1.FileName);
end;
```

Определите метод обработки события OnClick для кнопки BitBtn3:

```
procedure TForm4.BitBtn3Click (Sender: TObject);
begin
  Form1.MyTab.Close;  {Закроем таблицу, чтобы ее создавать}
  Form1.MyTab.ReadOnly:= False; {Разрешить редактирование}
  MemHelp.Clear;      {Очистим текстовое окно}
  Form4.Hide;         {Спрячем форму}
  Form4.Close;        {Удалим форму из памяти}
end;
```

6.6.3. Пример созданного приложением файла справки

```
Документальная справка про Соль
Артикул: 1002
Количество: 3
```

```
Цена: 10
Дополнения: Хлеб, соль и спички — предметы первой необходимости
для путешественника
(этот текст набран в окне с меткой «Справка:»)
Справка выдана: 01.05.200913:10:31
```

Библиографический список

1. Архангельский А. Я. Object Pascal в Delphi 5 / А. Я. Архангельский. М. : БИНОМ, 2002. 363 с.
2. Единая система программной документации. М. : Изд-во стандартов, 1982. 128 с.
3. Кибардин А. В. Основы информатики: в 2 ч. Ч. 1 / А. В. Кибардин, И. Г. Неудачин, В. И. Рогович. Екатеринбург, 2005. 204 с.
4. Форсайт Р. Паскаль для всех / Р. Форсайт. М. : Машиностроение, 1986. 288 с.
5. Стивенс Р. Delphi. Готовые алгоритмы / Р. Стивенс. М. : ДМК Пресс, 2001. 384 с.

Приложения

Приложение 1

Предопределенные обработчики исключительных ситуаций

Ниже вы найдете справочную информацию по предопределенным исключениям, необходимую для профессионального программирования в Delphi.

EAbort генерируется при вызове процедуры Abort. Предназначено для намеренного прерывания вычислений и быстрого выхода из глубоко вложенных процедур и функций.

EaccessViolation — ошибочный доступ к памяти; генерируется при попытке разыменования пустого указателя **nil**, попытке записи в кодовую страницу, доступа к адресу вне памяти, распределенной приложению.

EFCreateError указывает на ошибку при создании файла; например, недопустимое имя файла или указанный файл уже существует и не может быть перезаписан, т. к. пользователь не обладает соответствующим уровнем доступа. Происходит в случае ошибок создания потока (например, при некорректном задании файла потока).

EConvertError происходит в случае возникновения ошибки преобразования типов данных при выполнении функций StrToInt и StrToFloat, когда конвертация строки в соответствующий числовой тип невозможна.

EInOutError происходит при ошибках ввода-вывода при включенной директиве {$I+}. Ошибка файлового ввода-вывода. В поле ErrorCode экземпляра исключения содержится код ошибки ввода-вывода. Эта исключительная ситуация может возникнуть, только если программа скомпилирована с директивой {$I} (она используется по умолчанию). Коды ошибок следующие: 2 — файл не найден,

3 — неверное имя файла, 4 — слишком много открытых файлов, 5 — доступ запрещен, 100 — конец файла, 101 — диск переполнен и 106 — некорректный ввод. Также допустимы другие коды ошибок.

EDBEditError — ошибка при попытке приложения использовать данные, не соответствующие заданной маске для поля.

EDivByZero вызывается в случае деления на ноль, как результат RunTime Error 200.

EIntOverflow вызывается при попытке выполнения операций, приводящих к переполнению целых переменных, как результат RunTime Error 215 при включенной директиве {$Q+}.

ERangeError вызывается при попытке обращения к элементам массива по индексу, выходящему за пределы массива, как результат RunTime Error 201 при включенной директиве {$R+}.

EDatabaseError — ошибка работы с базами данных.

EInvalidGraphic вызывается при попытке передачи в LoadFromFile файла несовместимого графического формата.

EInvalidGraphicOperation вызывается при попытке выполнения операций, неприменимых для данного графического формата (например, Resize для TIcon).

EInvalidGridOperation генерируется при некорректных операциях с компонентами **Grid**, например при попытке обращения к несуществующей ячейке.

EInvalidPointer происходит при попытке освобождения уже освобожденного или еще неинициализированного указателя, при вызове Dispose (), FreeMem () или деструктора класса.

EListError вызывается при обращении к элементу наследника TList по индексу, выходящему за пределы допустимых значений (например, объект TStringList содержит только 10 строк, а происходит обращение к одиннадцатой).

EinvalidArgument — выходящее за допустимые рамки значение в специализированных математических и финансовых функциях из модуля Math.

Eprinter указывает на ошибку, возникшую во время печати.

EVariantError вызывается при неверной операции с типом variant.

EZeroDivide вызывается в результате деления на ноль.

EOverflow происходит как результат переполнения операций с плавающей точкой при слишком больших величинах. Соответствует RunTime Error 205.

EInvalidPointer происходит при попытке освобождения уже освобожденного или еще неинициализированного указателя, при вызове Dispose (), FreeMem () или деструктора класса.

EZeroDivide вызывается в результате деления на ноль.

EMenuError вызывается в случае любых ошибок при работе с пунктами меню для компонент TMenu, TMenuItem, TPopupMenu и их наследников.

EOutOfMemory происходит в случае вызовов New (), GetMem () или конструкторов классов при невозможности распределения памяти. Соответствует RunTime Error 203.

EPropertyError — ошибка при задании значения свойства.

ETreeViewError генерируется при использовании неверного индекса для компонента **TreeView**.

EStackOverflow происходит при ошибках работы со стеком. Генерируется, если стек текущего потока достиг последней страницы памяти, т. е. программе грозит нехватка памяти. Причиной этой проблемы обычно являются большие локальные переменные в процедурах и функциях, а также глубоко вложенные рекурсивные подпрограммы. (Переместите большие переменные в глобальную область).

EnoResultSet генерируется обьектом TQuery, если в запросе отсутствует оператор SELECT.

EWin32Error генерируется при возникновении ошибки Windows. Обработчиком по умолчанию выводится диалоговое окно с кодом ошибки и строкой сообщения. Чтобы получить сообщение операционной системы, можно использовать функцию Win32Check из модуля SysUtils.

EPrivilege генерируется при нарушении уровней привилегии процессора, например, если присваимаемое значение выходит за рамки допустимых или имеет некорректный тип данных.

EDBEditError — ошибка при попытке приложения использовать данные, не соответствующие заданной маске для поля.

EClassNotFound обычно происходит, когда в описании класса формы удалено поле-ссылка на компоненту, вставленную в форму в режиме дизайнера. Вызывается, в отличие от EResNotFound, в RunTime.

EInvalidImage вызывается при попытке чтения файла, не являющегося ресурсом, или разрушенного файла ресурса специализированными функциями чтения ресурсов (например, функцией ReadComponent).

EMethodNotFound — аналогично EClassNotFound, только при несоответствии методов, связанных с теми или иными обработчиками событий.

EReadError происходит в том случае, когда невозможно прочитать значение свойства или другого набора байт из потока (в том числе ресурса).

EFOpenError — ошибка открытия файла. Вызывается, когда тот или иной специфицированный поток не может быть открыт (например, когда поток не существует).

EStringListError происходит при ошибках работы с объектом TStringList (кроме ошибок, обрабатываемых TListError), например, при обращении к списку по неверному индексу.

Приложение 2

Примеры тем баз данных

№	Тема	№	Тема	№	Тема
1	Автомобили Audi.	28	Змеи.	55	Реки.
2	Состав продуктов питания.	29	Методы разработки алгоритмов.	56	Комплектующие ЭВМ.
3	Автоматизированные справочники.	30	Рейтинговая оценка знаний студентов.	57	Калории в продуктах питания.
4	Автомобили.	31	Рестораны.	58	Рецепты блюд.
5	Автомобильные шины.	32	Комнатные растения.	59	Компьютерные игры.
6	Автосалон.	33	Родословное дерево.	60	Сессия.
7	Адресная книга.	34	Самолеты.	61	Склады продукции.
8	Анкета.	35	Криптография.	62	Соревнования.
9	Архитектурные памятники.	36	Огнестрельное оружие.	63	Тарифы мобильной связи.
10	Мобильные телефоны.	37	Архиваторы.	64	Операции с графами.
11	Психологические тесты.	38	Транспортные маршруты.	65	Спортивные автомобили.
12	Библиография.	39	Музыка.	66	Спутники планет.
13	Библиотека.	40	Насекомые.	67	Страны Европы.

№	Тема	№	Тема	№	Тема
14	Биоритмы.	41	Научные труды.	68	Страны.
15	Ведомости заработной платы.	42	Программы контроля и обучения.	69	Студенческий отряд охраны права и порядка.
16	Видеокарты.	43	Овощи.	70	Студенты.
17	Видеотека.	44	Протоколы.	71	Фонотека.
18	Генератор отчетов.	45	Спорт.	72	Фильмотека.
19	Генераторы заданий и программ.	46	Ассортименты товаров магазинов.	73	Телефонная записная книжка.
20	География	47	Операции со списками.	74	Электронные таблицы.
21	Гороскопы.	48	Оружие.	75	Фармацевтика.
22	Государства.	49	CD-каталог.	76	Проект и анализ сети.
23	Графические, текстовые и прочие редакторы.	50	Интеллектуальные справочники, каталоги.	77	Тарифы междугородных телефонных переговоров.
24	Группа студентов.	51	Процессоры.	78	Хиромантия.
25	Диеты.	52	Метрика программ.	79	Чудеса.
26	Животные.	53	Расписания.	80	Музыкальный каталог.
27	Звезды.	54	Расходы.	81	Мебель.

Приложение 3

Файловые подпрограммы

Помните, что стандартные процедуры или функции можно вызывать, если в начале программы предложением **users** подключены модули, содержащие их описания. Вот список подпрограмм, отсортированных по категориям и по алфавиту. Информация по каждой подпрограмме содержит следующие поля:

Имя модуля с описанием, Подпрограмма (список параметров); Описание.

1. Подпрограммы управления файлами

System, procedure **AssignFile** (var F; FileName: string); Связывает файловую переменную с именем файла.

System, procedure **CloseFile** (var F); Закрывает файл по файловой переменной.

SysUtils, function **DeleteFile** (const FileName: string): Boolean; Удаляет файл.

SysUtils, procedure **FileClose** (Handle: Integer); Закрывает файл по дескриптору.

SysUtils, function **FileCreate** (const FileName: string): Integer; Создает новый файл.

Описание. FileCreate создает новый файл с указанным именем. Если возвращаемое значение положительно, функция выполнилась успешно, и возвращенное значение — дескриптор нового файла. Возвращаемое значение –1 указывает, что произошла ошибка.

Примечание. Использование несобственных Pascal дескрипторов файлов, как в FileCreate, не поощряется. Такие процедуры отображаются в функции Windows API и возвращают дескрипторы, а не обычные файловые переменные Pascal. Это низкоуровневые процедуры файлового доступа. Для обычных действий с файлами вместо них используют AssignFile, Rewrite, и Reset.

SysUtils, function **FileExists** (const FileName: string): Boolean; Проверяет наличие файла.

SysUtils, function **FileOpen** (const FileName: string; Mode: LongWord): Integer; Открывает заданный файл, используя указанный режим доступа.

Описание. Используйте FileOpen, чтобы открыть файл и получить дескриптор файла в Windows. Значение режима доступа построено осуществлением операции «OR» над одной из fmOpen констант с одной из fmShare констант, определенной в Fileopen константе режима доступа Mode. Если возвращаемое значение положительно, то функция применилась успешно, и ее значение — дескриптор открытого файла. Возвращаемое значение –1 указывает, что произошла ошибка.

Примечание. Использование несобственных Pascal дескрипторов файлов, как в FileOpen, не поощряется. Эти процедуры отображаются в функции Windows API и возвращают дескрипторы, а не нормальные файловые переменные Pascal. Они относятся к низкоуровневым процедурам файлового доступа. Для нормальных действий с файлами используют AssignFile, Rewrite, и Reset вместо этого.

SysUtils, function **FileRead** (Handle: Integer; var Buffer; Count: Integer): Integer; Читает из файла.

SysUtils, function **FileSearch** (const Name, DirList: string): string; Ищет файл в списке каталогов.

SysUtils, function **FileSeek** (Handle, Offset, Origin: Integer): Integer; Меняет позицию указателя.

SysUtils, function **FileWrite** (Handle: Integer; const Buffer; Count: Integer): Integer; Записывает в файл содержимое буфера в текущем положении указателя.

Описание. FileWrite записывает Count байт в файл, заданный дескриптором Handle из буфера, определенного как Buffer. Handle — дескриптор файла, возвращенный методом FileOpen или FileCreate. Возвращается фактически записанное число байтов или –1, если произошла ошибка.

Примечание. Не путайте подпрограммы, которые принимают или возвращают дескрипторы файлов, с теми, которые используют переменные файла Pascal (обычно обозначенные как переменная F). Чтобы записывать в файл, определенный переменной файла Pascal, используйте вместо этого Write, Writeln или BlockWrite.

SysUtils, procedure **FindClose** (var F: TSearchRec); Прекращает поиск файлов.

SysUtils, function **FindFirst** (const Path: string; Attr: Integer; var F: TSearchRec): Integer; Начинает поиск файлов.

SysUtils, function **FindNext** (var F: TSearchRec): Integer; Продолжает поиск файлов.

SysUtils, function **GetCurrentDir**: string; Определяет текущий каталог.

System, procedure **GetDir** (D: Byte; var S: string); Определяет текущий каталог.

SysUtils, function **SetCurrentDir** (const Dir: string): Boolean; Устанавливает текущий каталог.

2. Подпрограммы ввода-вывода

System, procedure **Append** (var F: Text); Добавляет текст в конец файла.

System, function **Eof** (var F): Boolean; Определяет конец файла.

System, function **FilePos** (var F): Longint; Определяет позицию указателя.

System, function **FileSize** (var F): Integer; Определяет размер файла.

System, function **IOResult**: Integer; Определяет ошибки предыдущего ввода/вывода.

System, procedure **Reset** (var F [: File; RecSize: Word]); Открывает файл.

System, procedure **Rewrite** (var F: File [; Recsize: Word]); Создает и открывает новый файл.

System, procedure **RmDir** (S: string); Удаляет каталог.

System, procedure **Seek** (var F; N: Longint); Устанавливает позицию указателя.

System, procedure **Truncate** (var F); Усекает файл до текущей позиции указателя.

3. Подпрограммы текстовых файлов

Printers, procedure **AssignPrn** (var F: Text); Связывает файловую переменную с принтером.

System, function **Eoln** [(var F: Text)]: Boolean; Определяет конец строки.

System, procedure **Erase** (var F); Удаляет файл.

System, procedure **Flush** (var F: Text); Переписывает данные в файл из его буфера.

System, procedure **Read** (F, V1 [, V2,.., Vn]); Читает из файла.

System, procedure **Readln** ([var F: Text;] V1 [, V2, ..., Vn]); Читает из файла до конца строки.

System, function **SeekEof** [(var F: Text)]: Boolean; Определяет конец файла.

System, function **SeekEoln** [(var F: Text)]: Boolean; Определяет конец строки.

System, procedure **SetTextBuf** (var F: Text; var Buf [; Size: Integer]); Устанавливает новый буфер.

System, procedure **Write** (F, V1 [, V2,.., Vn]); Записывает в файл.

System, procedure **Writeln** ([var F: Text;] V1 [, V2, ..., Vn]); Записывает в файл с концом строки.

Оглавление

Учебное издание

Неудачин Илья Георгиевич

ТАБЛИЦЫ DELPHI
ДЛЯ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Редактор Т. Е. Мерц
Верстка О. П. Игнатьевой

Подписано в печать 04.08.2016. Формат 70×100/16.
Бумага писчая. Печать цифровая. Гарнитура Newton.
Уч.-изд. л. 5,0. Усл. печ. л. 7,7. Тираж 50 экз.
Заказ 268

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: 8 (343)375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620075, Екатеринбург, ул. Тургенева, 4
Тел.: 8 (343) 350-56-64, 350-90-13
Факс: 8 (343) 358-93-06
E-mail: press-urfu@mail.ru


УНИВЕРСИТЕТСКОГО КНИГОИЗДАНИЯ
УНИВЕРСИТЕТСКАЯ КНИГА
2015


9 785799 617905